Interoperabilidade na prática: Aplicativo Delphi acessando um servidor WCF .NET

Clube Delphi

A evolução do LiveBindings
Entenda porque esse recurso é indispensável

Emissão de contratos e mala direta
Desenvolva um editor de textos completo

Delphi na Nuvem
Envie e baixe arquivos do Windows Azure usando Delphi

Edição 156 :: R$ 14,90

DEVMEDIA

INTEGRAÇÃO DE SISTEMAS

Explorando os recursos multiplataforma do Delphi XE5

ISSN 1517990-7

9 771517 990009 00156

CURSOS ONLINE

Clube Delphi

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

CONHEÇA ALGUNS DOS CURSOS

- Curso de Multicamadas com Delphi e DataSnap
- Delphi para Iniciantes
- Criando componente Boleto em Delphi
- Loja Virtual em Delphi Prism

Para mais informações :

www.devmedia.com.br/cursos/delphi

(21) 3382-5038

http://www.devmedia.com.br

DEVMEDIA

Clube Delphi

156ª Edição 2014 ISSN 1517990-7 Impresso no Brasil

Editor Geral

Paulo Quicoli (pauloquicoli@gmail.com)

Equipe Editorial

Guinther Pauli (guintherpauli@gmail.com)

Fabrício Hissao Kawata (fabricio.kawata@bol.com.br)

Giuliano Scombatti Pinto (giuliano@sygnux.com.br)

Daniel Sobrinho Laporte (daniel.laporte@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Consultor Técnico

Wesley Yamazack (yamazack@gmail.com)

Capa e Diagramação

Romulo Araujo

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Paulo Quicoli - Editor da Revista
pauloquicoli@gmail.com

Paulo Quicoli - **Editor Geral**

pauloquicoli@gmail.com
quicoli.wordpress.com
twitter.com/pauloquicoli

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DEVMEDIA

Sumário

Conteúdo sobre Boas Práticas, Conteúdo sobre Novidades, Artigo no estilo Solução Completa

04 – Delphi XE5: integração de sistemas móveis e desktops
[*Renan Sanches Saraiva dos Santos*]

Conteúdo sobre Novidades

16 – LiveBinding: a evolução no Delphi
[*Filipe Dalepiane*]

Artigo no estilo Solução Completa

27 – WPTools: edição, correção ortográfica e mala direta de textos no Delphi XE 3
[*Paulo Quicoli*]

Artigo no estilo Solução Completa

35 – Delphi WCF: aplicações multicamadas
[*Guinther Pauli e Jones Granatyr*]

43 – Delphi XE2: Cloud Computing e Windows Azure
[*Rafael Santos Pimenta*]

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Delphi XE5: integração de sistemas móveis e desktops

Aplicando Datasnap RESTFul e desenvolvimento Android

Atualmente está se vivenciando um momento de explosão da tecnologia móvel, onde os hardwares destes dispositivos já se equiparam aos de um desktop, seus sistemas operacionais estão cada vez mais evoluídos, robustos e intuitivos, trazendo novas funções a cada versão e prezando sempre pela comodidade e a facilidade de operar um computador na palma da mão.

Para que as informações sejam disponibilizadas a estes dispositivos, elas deverão permitir ser acessadas e manipuladas de forma prática e clara. Muitos dos sistemas de gestão das empresas, por exemplo, são desktop e funcionam apenas em rede local (arquitetura cliente-servidor). Neste cenário apresentado, surge o dilema de oferecer a solução de mobilidade para estes, sem exigir alto investimento e sem trocar as atuais infraestruturas e soluções existentes, que acabariam por inviabilizar o projeto pelo alto custo e tempo de desenvolvimento. A solução viável encontrada é a integração dos sistemas através de um Web Service que forneça serviços de acesso a base de dados do sistema desktop e disponibilize essas informações para serem consumidas pela mobilidade. É inevitável: os gestores de TI precisam aplicar cada vez mais tempo e recursos na integração de suas aplicações de negócios aos dispositivos móveis.

O objetivo deste artigo é a exploração do tema integração de sistemas móveis e Desktop através de um Web Service implementado com o DataSnap, bem como o estudo das tecnologias e arquiteturas envolvidas, além do desenvolvimento de um aplicativo Client Mobile no Delphi XE5 para consumir as rotinas do DataSnap.

Fique por Dentro

O artigo apresenta a utilização do DataSnap REST e um aplicativo móvel desenvolvido no Delphi XE5. O aplicativo Desktop representa um sistema ERP qualquer que necessita expor parte de suas funcionalidades para fora da rede interna de uma empresa. Isso é feito através da criação de serviços DataSnap REST que trafegam informações no formato JSON. Já o aplicativo móvel tem como alvo a plataforma Android, e para seu desenvolvimento são utilizados os recursos trazidos pelo Delphi XE5, garantindo que o aplicativo Android acesse os serviços DataSnap REST de uma forma simples e segura. Sendo assim, de uma forma geral serão demonstrados os conceitos e etapas necessárias para integrar sistemas Desktop com sistemas móveis.

Tipos de integração

Existem vários tipos de integração de sistemas de informação que podem ser classificados de acordo com os métodos e a tecnologia utilizada. Basicamente existem as seguintes abordagens para integração entre sistemas e organizações:

• **Orientada a dados:** Este tipo de integração envolve o processo de capturar informações de um banco de dados e inseri-las em outro, ou seja, a transferência de dados. Mesmo parecendo simples, este procedimento exige toda modelagem das regras de negócio, além da possibilidade de envolver grande quantidade de dados e diversidade de banco de dados. E as aplicações que suportam os sistemas de informação não são modificadas, permanecendo assim não integradas;

• **Orientada a Interface de Aplicações:** Neste tipo, a integração se baseia na exposição de mecanismos de acesso aos dados ou funções de negócio. A integração orientada a interface de apli-

cações tem sido adotada por pacotes de aplicações como da SAP, PeopleSoft e Baan. Formatos diferentes dessas interfaces dificultam a integração. Mesmo dentro da mesma empresa as interfaces de aplicações são necessárias, sendo utilizadas para obter informações, processá-las e transmiti-las em um formato que a aplicação destino compreenda;

• **Orientada a métodos:** Esta integração utiliza, além das interfaces, o compartilhamento das funcionalidades da aplicação, ou seja, visa utilizar um determinado método em diferentes sistemas, dentro ou fora da organização. Dentre as tecnologias envolvidas neste tipo de integração estão: Objetos Distribuídos, Servidores de Aplicação e Processamento de Transações;

• **Orientado a Interface de Usuário:** Neste tipo de integração, a interface do usuário representa funcionalidades e informações pertencentes a diferentes organizações, unificando sistemas distintos apenas no front-end e diminuindo a complexidade e custo da integração. Normalmente na Web esta integração é estabelecida por uma estrutura denominada portal. Um exemplo de integração no nível de interface do usuário é a compra em um site de e-commerce, onde o pagamento muitas vezes é feito com cartão de crédito e a autorização da compra no cartão pode estar diretamente ligada ao sistema da operadora do cartão de crédito;

• **Orientado a Processos de Negócios:** Este tipo estabelece um mapeamento dos processos de negócio e fluxo das informações das organizações envolvidas. Define uma camada de abstração dos processos no topo das aplicações e tecnologias utilizadas na integração. Está relacionado com elementos como pessoas, empresas e pedidos;

• **Orientada a Serviços:** Similar a orientada a métodos, este tipo de integração tem por objetivo expor funcionalidades de negócios, definidas no contexto de uma organização, através de serviços que possam ser acessados por outras aplicações através de uma rede. Utiliza como base a arquitetura orientada a serviços - Service Oriented Architecture (SOA), onde clientes e provedores visam se interagir dinamicamente para disponibilizar e consumir serviços. Este será o tipo de integração utilizada neste trabalho.

Arquitetura Orientada a Serviços(SOA)

No contexto onde as empresas precisam sempre estar prontas para as constantes mudanças de mercado, principalmente no que se refere à manutenção e aperfeiçoamento das infraestruturas de TI, o SOA vem de encontro a estas necessidades, já que visa implementar processos de negócios da empresa, oferecendo funções flexíveis, reutilizáveis, padronizadas que são denominadas serviços.

SOA não é uma tecnologia nem um produto, mas sim um estilo de projeto que visa pensar e planejar a integração através dos processos de negócio disponibilizados por meio de serviços, tanto é que pode ser abordada do ponto de vista do negócio e do ponto de vista da TI.

Atualmente o principal meio de integração de sistemas se dá através da SOA, que vai além de apenas possibilitar o intercâmbio de informações com sistemas antigos, mas através da arquitetura orientada a serviço é possível integrar sistemas com plataformas diferentes (Mobile, Web, Desktop) de maneira simples, sem a exigência de grandes custos e também sem consumir muito tempo de desenvolvimento. Constatou-se que através da SOA o conceito de sistemas monolíticos desaparece, dando origem a ideia de processos de negócio que podem ser facilmente alterados e tranquilamente disponibilizados para utilização em outros meios, projetos ou sistemas.

SOA e os Web Services

A princípio, deve-se ter em mente que SOA é um modelo arquitetônico que não depende de plataforma de tecnologia e que Web Services é uma das tecnologias que se pode utilizar para implementar SOA, embora este tenha surgido antes. Existem outras tecnologias mais antigas que também podem ser utilizadas como CORBA, DCOM, RPC, mas não vem ao contexto deste artigo.

O sucesso dos Web Services se deu bem antes da computação orientada a serviços, devido necessidade de facilitar a integração das aplicações distribuídas que existiam antigamente. Devido esta ferramenta de integração ter aparecido primeiro, ela influenciou em várias características da arquitetura orientada a serviços, contudo, só depois da consolidação do conceito de SOA é que foi possível tirar um proveito maior sobre a interoperabilidade aberta pelos Web Services.

Embora se tenha uma visão simplista de que SOA é Web Services e vice e versa, o fundamento dos dois somente tem similaridade na questão da integração, pois a arquitetura orientada a serviço vai desde a estruturação dos processos de negócios por meio de uma estratégia de TI da empresa até sua implementação em si através de uma tecnologia, e pode-se considerar que os Web Services são o último passo do resultado prático do SOA, quando tudo está pronto para disponibilizar serviços a fim de serem consumidos ou integrar sistemas.

Web Services

Um Web Service é um componente de Software que permite a interação entre aplicações diferentes e heterogêneas pela rede. E os dois principais tipos de utilização é através do protocolo SOAP e REST. O SOAP é um protocolo utilizado há algum tempo, no qual ele empacota a mensagem e transmite geralmente através do protocolo HTTP com serialização de arquivos XML. Já o REST é uma abordagem mais atual e será a forma utilizada no exemplo prático deste artigo.

O termo REST (Representational State Transfer) surgiu a partir da tese de doutorado de Roy Fielding (também um dos criadores do protocolo HTTP) publicada no ano de 2000 e define um novo padrão para descrever qualquer interface que envie informações de um domínio específico através de HTTP sem nenhuma camada adicional.

Web Service RESTful também é uma forma de implementar o SOA e o seu funcionamento e criação é bem mais simples em comparação com os Web Services padrões SOAP, já que não necessita de arquivo descritor como é o WSDL, não envolve envelopamento e somente utiliza os recursos puros do protocolo HTTP.

Outra característica desse estilo é a possibilidade de utilizar arquivos JSON (JavaScript Object Notation) ao invés de XML. JSON é um texto que representa um objeto sendo formatado da seguinte maneira: {texto: valor, texto: valor, ... texto: valor}. Assim, é possível representar uma escrita complexa de informações de uma forma mais simples, leve e rápida em comparação com o XML. E se tratando de aplicações móveis no qual geralmente utilizam as redes de dados móveis ao invés de banda larga para trocarem informações na Internet, é fundamental a necessidade de enxugar os dados que serão transferidos entre os sistemas, portanto a utilização de JSON se mostra mais viável.

De fato, o modo de implementar um Web Service baseado em REST veio para facilitar e melhorar a disponibilização de serviços entre as organizações principalmente pela sua simplicidade.

O Projeto de Integração

Vamos criar um projeto extremamente simples tanto no Desktop quanto Mobile para podermos demonstrar na prática os conceitos de integração entre plataformas. O projeto será de um cadastro de contatos, onde terá somente uma tabela em um banco de dados Firebird, assim como mostrado na **Listagem 1**.

Para deixarmos o projeto de forma organizada, definiremos a estrutura de pastas separadas por projeto, dentro do diretório ProjetoIntegração, como ilustra a **Figura 1**. O banco de dados ficará localizado na raiz da pasta principal.

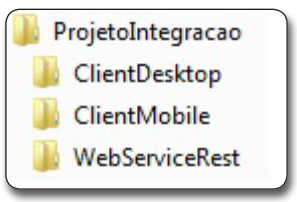

Figura 1. Diretórios do projeto

Listagem 1. Tabela de Contato

```
CREATE TABLE CONTATO (
   ID          INTEGER NOT NULL,
   NOME        VARCHAR(50),
   EMAIL       VARCHAR(50),
   TELEFONE    VARCHAR(20)
);
ALTER TABLE CONTATO ADD CONSTRAINT PK_CONTATO PRIMARY KEY (ID);
CREATE SEQUENCE GEN_ID_CONTATO;
```

O projeto Desktop

O projeto Desktop terá dois formulários apenas, o DataModule e o formulário de contatos. No DataModule vamos adicionar os componentes básicos de conexão, SQLConnection, SQLDataSet, ClientDataSet e DataSetProvider e configurar as ligações e a conexão. Agora adicionaremos os Fields no SQLDataSet e ClientDataSet para em seguida conectá-los aos DBEdits. No formulário de contatos serão adicionados os componentes necessários para montar a interface da aplicação, conforme a **Figura 2,** e um DataSource para ligar nos componentes de acesso ao banco.

A **Listagem 2** permite identificar os tipos e nomes dos componentes da tela.

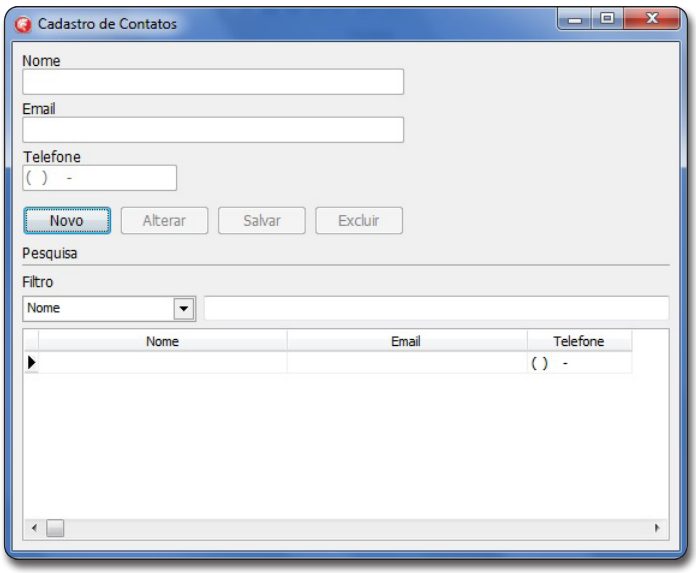

Figura 2. Interface do Cadastro de Contatos

Listagem 2. Formulário de Contato

```
01 unit UCadContato;
02 interface
03 uses
04  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
    System.Classes, Vcl.Graphics,  Vcl.Controls, Vcl.Forms, Vcl.Dialogs,
    Vcl.StdCtrls, Vcl.Mask, Vcl.DBCtrls, Data.DB, Vcl.Grids, Vcl.DBGrids,
    Vcl.ExtCtrls;
05  type
06  TFCadContato = class(TForm)
07    lblNome: TLabel;
08    EdtNome: TDBEdit;
09    lblEmail: TLabel;
10    EdtEmail: TDBEdit;
11    lblTelefone: TLabel;
12    EdtTelefone: TDBEdit;
13    DSContato: TDataSource;
14    BtnNovo: TButton;
15    BtnAlterar: TButton;
16    BtnSalvar: TButton;
17    BtnExcluir: TButton;
18    GridPesquisa: TDBGrid;
18    Bevel1: TBevel;
19    lblPesquisa: TLabel;
20    EdtPesquisa: TEdit;
21    EdtFiltro: TComboBox;
22    lblFiltro: TLabel;
23    procedure DSContatoStateChange(Sender: TObject);
24    procedure BtnNovoClick(Sender: TObject);
25    procedure BtnAlterarClick(Sender: TObject);
26    procedure BtnSalvarClick(Sender: TObject);
27    procedure BtnExcluirClick(Sender: TObject);
28    procedure FormShow(Sender: TObject);
28    procedure HabilitaDesabilitaCampos;
29    procedure EdtPesquisaChange(Sender: TObject);
30  private
31    { Private declarations }
32  public
33    { Public declarations }
34  end;
```

Na mesma tela o usuário poderá incluir/alterar/excluir/consultar os contatos, e os métodos necessários para permitir estas funcionalidades são os demonstrados na **Listagem 3**.

O método Alterar (linhas 01 a 05) simplesmente edita o ClientDataSet CDSContato, colocando o foco no controle EdtNome respectivamente. O método excluir (linhas 07 a 12) simplesmente deleta os registros do CDSContato, persistindo-os em seguida e chamando o procedimento HabilitaDesabilitaCampos (mostrado a seguir na mesma listagem). O método Novo (linhas 14 a 19) apenas limpa o edit de Pesquisa, insere um novo registro no CDSContato e coloca o foco no componente edtNome. O método Salvar tem praticamente a mesma funcionalidade do método Excluir, porém, neste caso apenas é persistido um registro depois de modificado ou inserido. A seguir, entre as linhas 27 e 30 temos a implementação do evento OnChangeState do DataSource, que ocorrerá sempre que alguma operação de CRUD for realizada. Esta simplesmente chamará o método HabilitaDesabilitaCampos que, como o próprio nome sugere, fará a visibilidade dos campos conforme o status do DataSet. Entre as linhas 32 a 47 há o evento OnChange do Edit de pesquisa. Observe que entre as linhas 36 a 42 é realizado um teste com o ItemIndex do ComboBox, testando se o valor selecionado é um Nome, E-mail ou Telefone, fazendo em seguida a concatenação das instruções SQL de acordo com a variável Where. A instrução SQL é repassada ao ClientDataSset CDSContato que simplesmente a executa, exibindo os dados encontrados.

O evento OnShow do formulário (linhas 49 a 52) simplesmente abre o DataSet CDSContato. Por último, entre as linhas 54 e 63 os controles Edit são habilitados/desabilitados conforme o estado do DataSet (CDSContato).

Desenvolvimento do servidor DataSnap REST

Presente desde a versão 2010 do Delphi, os servidores DataSnap RESTful permitem trafegar dados no formato JSON, o que facilita a publicação de serviços na Web para que o mesmo seja consumido através do protocolo HTTP sem distinção de plataforma ou linguagem do cliente.

Para o exemplo prático do artigo também vamos utilizar o Delphi XE5 tanto para criar o DataSnap Server, quanto o aplicativo Android Client. Iniciando o desenvolvimento, primeiramente criaremos um projeto dentro do próprio grupo do projeto Desktop para ficar mais bem organizado, do tipo DataSnap REST Application através do próprio Wizard do Delphi, que automaticamente já configura todos os componentes e formulários necessários para a implementação de um servidor REST. É importante durante os passos de criação, marcar o checkbox Authorization para que o Delphi já adicione o componente e os códigos necessários para a autenticação no servidor, já que os métodos expostos por REST podem ser acessados através de links no navegador, portanto se não adicionar uma autenticação fica totalmente aberto o acesso aos serviços disponibilizados pelo DataSnap. Também é necessário

Listagem 3. Métodos do sistema Desktop

```
01 procedure TFCadContato.BtnAlterarClick(Sender: TObject);
02 begin
03   FDataModule.CDSContato.Edit;
04   EdtNome.SetFocus;
05 end;
06
07 procedure TFCadContato.BtnExcluirClick(Sender: TObject);
08 begin
09   FDataModule.CDSContato.Delete;
10   FDataModule.CDSContato.ApplyUpdates(0);
11   HabilitaDesabilitaCampos;
12 end;
13
14 procedure TFCadContato.BtnNovoClick(Sender: TObject);
15 begin
16   EdtPesquisa.Clear;
17   FDataModule.CDSContato.Append;
18   EdtNome.SetFocus;
19 end;
20
21 procedure TFCadContato.BtnSalvarClick(Sender: TObject);
22 begin
23   FDataModule.CDSContato.Post;
24   FDataModule.CDSContato.ApplyUpdates(0);
25 end;
26
27 procedure TFCadContato.DSContatoStateChange(Sender: TObject);
28 begin
29   HabilitaDesabilitaCampos;
30 end;
31
32 procedure TFCadContato.EdtPesquisaChange(Sender: TObject);
33 Var
34   where: String;
35 begin
36   case EdtFiltro.ItemIndex of
37    //Nome
38     0: where:='where Nome like '+QuotedStr('%'+EdtPesquisa.Text+'%');
39    //Email
40     1: where:='where Email like '+QuotedStr('%'+EdtPesquisa.Text+'%');
41    //Telefone
42     2: where:='where Telefone like '+QuotedStr('%'+EdtPesquisa.Text+'%');
43   end;
44   FDataModule.CDSContato.Close;
45   FDataModule.CDSContato.CommandText := 'Select * from Contato '+where;
46   FDataModule.CDSContato.Open;
47 end;
48
49 procedure TFCadContato.FormShow(Sender: TObject);
50 begin
51   FDataModule.CDSContato.Open;
52 end;
53
54 procedure TFCadContato.HabilitaDesabilitaCampos;
55 begin
56   EdtNome.Enabled    := FDataModule.CDSContato.State in
                  [dsEdit,dsInsert];
57   EdtEmail.Enabled   := EdtNome.Enabled;
58   EdtTelefone.Enabled := EdtNome.Enabled;
59   BtnAlterar.Enabled  := (FDataModule.CDSContato.State in [dsBrowse]) and
60    (FDataModule.CDSContato.RecordCount>0);
61   BtnExcluir.Enabled  := BtnAlterar.Enabled;
62   BtnSalvar.Enabled   := EdtNome.Enabled;
63 end;
```

modificar a classe ancestral para os métodos do servidor para TDSServerModule, já que o projeto de integração envolve o acesso a banco de dados e DataSets. Após o término dos passos, tem-se a tela básica do servidor como mostrado na **Figura 3** e também a Unit ServerMethodsUnit1 onde se desenvolve os métodos.

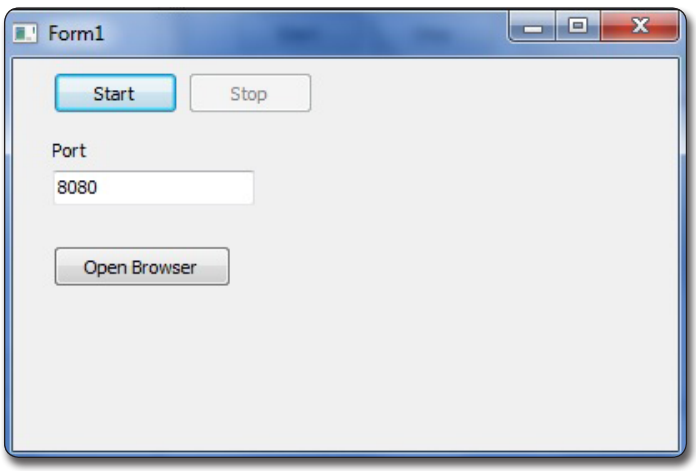

Figura 3. Datasnap básico criado pelo wizard

Depois disso, é importante adicionar um DataModule ao projeto para os componentes de acesso ao banco, pois, se adicionarmos ao ServerMethods1, eles são criados e destruídos a cada chamada de método. No novo DataModule é adicionado o componente de conexão SQLConnection e um SQLDataset.

Partindo do princípio de que muitas das vezes que formos integrar sistemas móveis e Desktop, o projeto Desktop quase sempre já existirá, é importante utilizar as mesmas regras de negócio necessárias para manter o funcionamento idêntico do sistema nas outras plataformas. Portanto, o próximo passo é escrever os métodos necessários para transpor as funcionalidades do sistema Desktop no Mobile, lembrando que todos os métodos resultam em JSON, inclusive as funções declaradas para retornar STRING ou REAL, o resultado sempre será JSON. Na **Listagem 4** podemos ver o método responsável por buscar no banco todos os contatos e retorna-los através de JSON. E através da rotina da **Listagem 5** podemos receber um Objeto JSON contato, e inseri-lo no banco. Basicamente estes são os métodos essenciais para o funcionamento deste sistema de Cadastro de Contatos selecionados para serem desenvolvidas no cliente Mobile.

Na **Listagem 4**, entre as lilnhas 05 a 07, a query realiza a busca de contatos cadastrados e na linha 08 é criado um JSONArray. Entre as linhas 09 a 16, é criado um JsonObject dentro de um loop na query, a partir da linha 12 os dados são adicionados por meio do método AddPair(Chave, Valor). Isto fará com que sejam criados os registros em formato JSon.

Na **Listagem 5** é realizado o procedimento inverso da **Listagem 4**. Observe que o método UpdateInsereContato recebe um JSONObject como parâmetro de entrada. A seguir, entre as linhas 07 a 16 é realizada a operação de desserialização do JSON, onde os valores obtidos são passados as variáveis do tipo String.

Listagem 4. Método de listagem dos Contatos cadastrados

```
01 function TServerMethods1.listaContato: TJSonArray;
02 Var
03  Obj: TJsonObject;
04 begin
05  FDataModule.QrContato.Close;
06  FDataModule.QrContato.CommandText:=('Select * from Contato');
07  FDataModule.QrContato.Open;
08  result:=TJSONArray.Create;
09  while not FDataModule.QrContato.Eof do
10  begin
11   OBj:=TJsonObject.Create;
12   Obj.AddPair('Nome',TJSONString.Create(
     FDataModule.QrContato.FieldByName('Nome').AsString));
13   Obj.AddPair('Email',TJSONString.Create(
     FDataModule.QrContato.FieldByName('Email').AsString));
14   Obj.AddPair('Telefone',TJSONString.Create(
     FDataModule.QrContato.FieldByName('Telefone').AsString));
15   Result.AddElement(obj);
16   FDataModule.QrContato.Next;
17  end;
18 end;
```

Listagem 5. Método de Inclusão de Contato

```
01 function TServerMethods1.updateInsereContato(Contato: TJsonOBject)
   : String;
02 Var
03  Nome, Email, Telefone, xSql: String;
04  ID: Integer;
05  Par: TJSONPair;
06 begi
07  Par := TJSONPair.Create;
08  //Nome
09  Par := Contato.Get(0);
10  Nome:=(Par.JsonValue.Value);
11  //Email
12  Par := Contato.Get(1);
13  Email:=Par.JsonValue.Value;
14  //Telefone
15  Par := Contato.Get(2);
16  Telefone:=Par.JsonValue.Value;
17  //GENERATOR
18  FDataModule.QrContato.Close;
19  FDataModule.QrContato.CommandText:=(
    'SELECT GEN_ID(GEN_ID_CONTATO,1) FROM RDB$DATABASE');
21  FDataModule.QrContato.Open;
22  ID := FDataModule.QrContato.Fields[0].Value;
23
24  xSql:='Insert into Contato'
25      +'  (ID, Nome, Email, Telefone)'
26      +'  values ('+IntToStr(ID)+','
27      +QuotedStr(Nome)+','
28      +QuotedStr(Email)+','
29      +QuotedStr(Telefone)
30      +')';
31  Try
32   //INSERE CONTATO
33   FDataModule.QrContato.Close;
34   FDataModule.QrContato.CommandText:=xSQL;
35   FDataModule.QrContato.ExecSQL();
36   result:=('OK!');
37  Except on E:Exception Do
38   Begin
40    result:=('ERRO - '+slinebreak+'Erro: '+E.Message);
41   End;
42  End;
43 end;
```

Na linha 19 é disparado um Generator para obter o valor do ID. Nas linhas 24 a 30 a instrução SQL é montada, para posteriormente ser executada pelo QrContato (linha 35). Caso a inserção seja bem sucedida, é retornado um comando OK (linha 36), caso contrário, é retornado um erro (linha 40).

Já que foi selecionada a opção de autenticação, é importante adicionar uma validação de usuário e senha no evento DSAuthenticationManager1UserAuthenticate do componente DSAuthenticationManager que fica no formulário WebModule1, como mostra a **Listagem 6**. O código é bem simples como pode se notar. A variável Valid do parâmetro simplesmente retornará True (autenticado) caso o usuário e senha correspondam ao valor 'devmedia'.

Listagem 6. Método de autenticação do usuário no servidor.

```
procedure TWebModule1.DSAuthenticationManager1UserAuthenticate(
  Sender: TObject; const Protocol, Context, User, Password: string;
  var valid: Boolean; UserRoles: TStrings);
begin
   valid := (User='devmedia') and (Password='devmedia');
end;
```

Até agora desenvolvemos o cliente Desktop e o DataSnap server com os métodos necessários para expor as funcionalidades do sistema, a fim de serem consumidos em outra plataforma.

Aplicativo cliente Android

O mercado atual de aplicações está voltado para um novo paradigma de desenvolvimento após a explosão do advento aos dispositivos móveis. Atualmente a maioria das empresas de software está migrando ou já cogita migrar para a plataforma móvel, sendo que muitas aplicações web e sistemas corporativos consolidados já possuem seu módulo móvel interagindo paralelamente com a aplicação primária.

Esta nova abordagem de desenvolvimento trouxe também algumas características a serem levadas em conta na hora de desenvolver:

- Conexões de rede, como Wifi, bluetooth e 3G, nem sempre estão disponíveis o tempo todo já que a rede móvel não atinge todos os lugares e as redes fixas necessitam de roteadores próximos.
- Limitação do espaço de armazenamento, se comparado com o espaço de computadores, a quantidade possível para armazenar os dados é bem inferior.
- Processamento também inferior ao dos desktops, além disso, alta carga de uso do processador consome muita bateria.
- Tempo de bateria, este acaba por limitar o tempo de uso do dispositivo.

Alguns dos erros mais cometidos ao desenvolver uma aplicação móvel:

1. Tentar passar a mesma experiência de um Desktop a um dispositivo móvel: os dispositivos não tem o mesmo tamanho de tela, teclado externo, cache na memória entre outras coisas que apenas os desktops têm, ou seja, não contam com a mesma conveniência dos PCs. E ainda existem programadores que tentam construir funcionalidades extremamente complexas na pequena tela do dispositivo, o que acaba prejudicando a experiência do usuário e consumindo recursos do dispositivo. **Como evitar**: passe para a versão móvel somente as rotinas da aplicação desktop com maior probabilidade de uso, além de se preocupar em minimizar ao máximo a quantidade de toques na tela necessários para manipular o sistema.

2. Não priorizar significativamente a experiência do usuário: Deixe a aplicação o mais fácil possível de usar para que os usuários não tenham experiências negativas. **Como evitar**: o foco deve sempre estar no design de uma boa experiência com o usuário. Uma forma de facilitar a percepção do grau de aceitação do usuário é utilizando protótipos antes do lançamento da aplicação.

3. Permitir que a aplicação rode sobre outra camada sem ser diretamente no Sistema Operacional: É preciso reduzir cada milissegundo possível de tempo de resposta sem abdicar de utilizar os recursos mais importantes do dispositivo. Existem plataformas de desenvolvimento que abstrai detalhes dos dispositivos oferecendo o mínimo das funcionalidades comuns entre eles. **Como evitar:** a utilização de aplicativos verdadeiramente nativos a cada

Sistema Operacional otimiza a utilização do hardware tornando a aplicação mais fluída.

4. Não se preocupar com a segurança: É necessário medidas adicionais que protejam as informações que estão armazenadas no dispositivo ou trafegando na rede e não utilizar somente a segurança própria do aparelho. **Como evitar:** medidas de seguranças relacionadas à criptografia de dados devem sempre ser utilizadas.

Após breve embasamento sobre as peculiaridades do desenvolvimento Mobile, vamos à demonstração do projeto cliente Android no Delphi XE5 que irá se comunicar com os serviços disponibilizados pelo DataSnap. Dando início ao desenvolvimento móvel, no mesmo grupo dos projetos vamos adicionar um novo projeto, na parte de Mobile Projects e selecionar Blank Application, resultando no formulário de um celular Google Nexus 4. Aqui no meu caso, estou compilando para o Galaxy SII então preciso trocar o layout, mas cada um pode escolher o design que quiser. Agora colocamos um Toolbar alinhado ao topo e um Label dentro com o nome da aplicação. Uma das grandes diferenças do desenvolvimento Desktop para o Mobile no Delphi é que não temos formulários distintos para as telas, no lugar, trabalhamos somente com abas. Portanto vamos adicionar um TabControl alinhado em todo formulário, e três TabItems nomeandoos respectivamente para AbaMenu, AbaNovo e AbaPesquisa. É importante modificar a propriedade TabPosition do TabControl para tpNone para que as abas sejam acessadas somente via código através de botões. Na aba menu, vamos adicionar três botões assim como mostra a **Figura 4.**

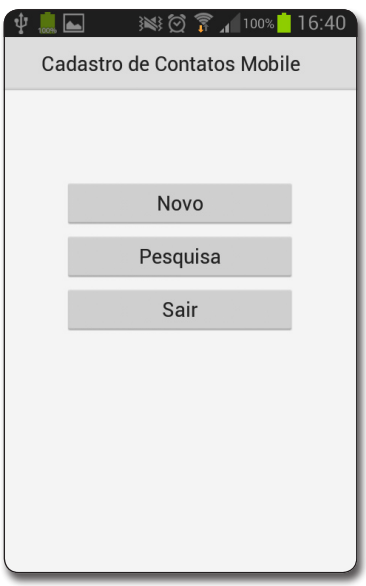

Figura 4. Menu do aplicativo móvel

Agora na AbaNovo, adicionamos um ToolBar alinhado embaixo com dois botões dentro (um a direita e o outro a esquerda), três Edits e três Labels, resultando na interface demonstrada na **Figura 5.**

Figura 5. Cadastro de Contatos Mobile

Na Aba Pesquisa, para que o layout fique igual ao da **Figura 6,** adicionaremos um GroupBox alinhado ao topo com três RadioButtons dentro, um ToolBar com um botão alinhado à esquerda e um ListView alClient. É importante marcar a opção SearchVisible do ListView para True, para que possamos pesquisar os resultados dentro do componente e modificar a propriedade ItemAppearence para MultiDetailItem (**ver BOX 1**), pois, vamos trabalhar com três linhas de dados.

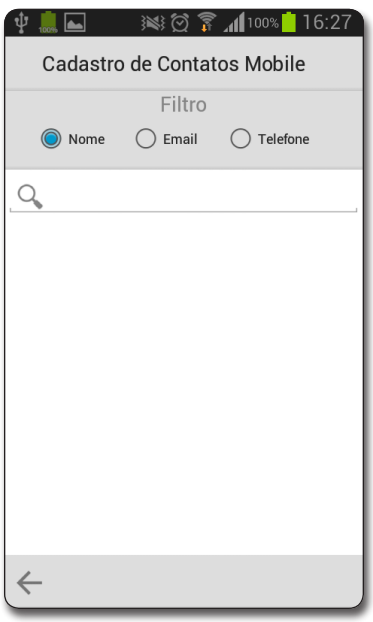

Figura 6. Pesquisa de Contatos Mobile

O layout da aplicação é este e agora partindo para o foco da integração dos Sistemas, para comunicar com o servidor DataSnap REST, o Delphi até possui componentes nativos para acesso direto, mas será utilizado o componente do Indy idHTTP,

VOCÊ TAMBÉM
ESTÁ BOIANDO
NESTA SOPA?
CSS
JPA
TDD
JSON
HTML
MYSQL

que faz as requisições pela URL, através dos métodos POST, PUT, DELETE, GET. Vamos adicionar um ao nosso projeto, e como configuramos a autenticação do usuário para poder acessar os serviços, primeiramente vamos configurar o usuário e senha enviados ao servidor pelo componente, caso contrário não será possível acessar os métodos. É interessante também configurar o tempo de resposta da requisição no componente idHTTP. A **Listagem 7** ilustra a configuração do idHttp.

Após isso, na AbaMenu, configuramos os acessos às abas através dos eventos OnClick dos botões, conforme a **Listagem 8.**

BOX 1. ListView MultiDetailItem

O ListView possui alguns formatos de exibição dos dados que podem ser configurados pela propriedade ItemAppearence. Neste exemplo estamos utilizando um formato que não vem por padrão instalado, que é a aparência MultiDetailItem. O package que acrescenta esta opção vem nos exemplos do próprio Delphi, localizado na pasta C:\Users\Public\Documents\RAD Studio\12.0\Samples\FireMonkeyMobile\ListView. Basta instalar o package e adicionar o caminho na Library.

Listagem 7. Configuração de usuário/senha e tempo de resposta da requisição

```
01 procedure TFMenu.FormShow(Sender: TObject);
02 begin
03  idHTTP1.Request.Username:='devmedia';
04  idHTTP1.Request.Password:='devmedia';
05  idHTTp1.ReadTimeout  :=5000;
06  idHTTp1.ConnectTimeout:=5000;
07  Abas.ActiveTab:=AbaMenu;
08 end;
```

Listagem 8. OnClick dos botões do Menu.

```
01 procedure TFMenu.BtnNovoClick(Sender: TObject);
02 begin
03  LimpaCamposContato;
04  Abas.ActiveTab:=AbaNovo;
05 end;
06
07 procedure TFMenu.BtnPesquisaClick(Sender: TObject);
08 begin
09  RBtnNome.IsChecked:=True;
10  PesquisaContato(0);
11  Abas.ActiveTab:=AbaPesquisa;
12 end;
13
14 procedure TFMenu.BtnSairClick(Sender: TObject);
15 begin
16  MainActivity.finish;
17 end;
18
19 procedure TFMenu.LimpaCamposContato;
20 begin
21  EdtNome.Text  :='';
22  EdtEmail.Text  :='';
23  EdtTelefone.Text:='';
24 end;
```

Na AbaNovo, para efeito de formatação, é possível configurar a propriedade KeyBoardType do EdtTelefone para vktNamePhonePad, que faz o dispositivo mostrar direto o teclado numérico quando focalizar no campo. Vamos configurar o evento OnClick do botão voltar e do botão confirmar desta aba conforme a **Listagem 9**. Ao preencher os dados e clicar em confirmar contato, todas as informações necessárias para o cadastro do contato são transformadas em um objeto JSON e enviadas por POST (**ver BOX 2**), para o método updateInsereContato existente no servidor que retorna se a operação foi bem sucedida. É importante atentar, que ao enviar um objeto JSON por POST (**BOX 3**) pelo idHTTP, é necessário transformar em um TStringStream para que a string JSON seja enviada da maneira correta pelo componente.

BOX 2. POST HTTP

Ao enviar uma requisição POST, PUT e DELETE são adicionados prefixos no nome do método pelo HTTP, desta forma sucessivamente: 'update', 'accept', 'cancel'. Ou seja, no servidor é preciso escrever os métodos também com esses prefixos para que eles sejam encontrados quando houver a requisição.

BOX 3. Envio de parâmetro String pelo servidor

Quando for necessário enviar um parâmetro para o servidor que não é suportado por GET ou DELETE (suportam string, número, booleano ou nulo), ou se o parâmetro estourar o limite da URL HTTP é preciso utilizar POST ou PUT na requisição.

Listagem 9. Envio das informações do contato ao servidor

```
01 procedure TFMenu.BtnConfirmaContatoClick(Sender: TObject);
02 Var
03  Contato: TJsonObject;
04  IJSO : String;
05  jSonToSend: TStringStream;
06  IResponse: String;
07 begin
08  if EdtNome.Text = '' then
09   ShowMessage('Preencha o Nome.')
10  Else
11  Begin
12   Contato:=TJsonObject.Create;
13   Contato.AddPair('Nome',   TJSONString.Create(EdtNome.Text));
14   Contato.AddPair('Email',  TJSONString.Create(EdtEmail.Text));
15   Contato.AddPair('Telefone',TJSONString.Create(EdtTelefone.Text));
16
17   IJSO     := Contato.ToString;
18   jSonToSend := TStringStream.Create(IJSO,TEncoding.UTF8);
19   IdHTTP1.Request.ContentType := 'application/json';
20   IResponse := IdHTTP1.Post(URL+'insereContato',jSonToSend);
21   jSonToSend.Free;
22
23   if Copy(IResponse,13,3)='OK!' then
24   Begin
25    ShowMessage('Contato cadastrado com sucesso!');
26    LimpaCamposContato;
27   End
28   Else
29   Begin
30    ShowMessage('Erro ao salvar, tente novamente'
31     +slinebreak+'Erro: '+Copy(IResponse,11,length(IResponse)));
32   End;
33  End;
34 end;
35 procedure TFMenu.BtrnVoltarAbaContatoClick(Sender: TObject);
36 begin
37  Abas.ActiveTab:=AbaMenu;
38 end;
```

ASSIM, O MERCADO TE ENGOLE!

PARAR DE SE ATUALIZAR PODE SER FATAL!

O mercado muda a todo momento e você precisa estar atento sempre. Seja um assinante MVP e devore tudo o que há de mais relevante no mercado de TI.

Invista na sua carreira!

TENHA ACESSO A:

- +DE 260 CURSOS ONLINE
- 09 REVISTAS MENSAIS
- 7.850 VÍDEO-AULAS

POR APENAS 59,90 MENSAIS

ASSINATURA MVP
MOST VALUABLE PROFESSIONAL

QUEM TEM ESTÁ TRANQUILO.

DEVMEDIA

Acesse: www.devmedia.com.br/mvp

Na **Listagem 9**, nota-se que nas linhas 08 e 09 é realizada uma validação no preenchimento do Nome, exibindo uma mensagem caso esteja em branco. Caso esteja preenchido, é criado um JsonObject que receberá os valores dos respectivos Edits (linhas 12 a 15). Na linha 18 a String contendo o JSON criado anteriormente é convertida em um StringStream para posteriorment ser enviado ao servidor (linha 20). Na linha 23 é checada a resposta vinda do server. Caso seja OK, automaticamente é exibida uma mensagem de contato cadastrado e os valores dos campos são esvaziados automaticamente. Caso contrário, é mostrada uma mensagem de erro (linha 30).

Partindo para a pesquisa, ao clicar no botão pesquisar da AbaMenu, o aplicativo já abrirá os contatos cadastrados chamando a rotina da **Listagem 10,** onde através do idHTTP, fazemos uma requisição ao serviço "listaContato" presente no servidor através da URL. Primeiramente criaremos uma constante chamada URL com o endereço do servidor de aplicação, como o exemplo a seguir:

```
const
  URL  = 'http://192.168.0.7:8080/datasnap/rest/TServerMethods1/';
```

O endereço de IP da rede interna é 192.168.0.7. O resultado sempre virá em forma de JSON e neste caso é um vetor de objetos que precisa ser tratado. O Delphi por padrão retorna duas chaves a mais na string JSON, o que deve ser contornado com um StringReplace para que seja possível interpretar o vetor de objetos. Feito isso, os resultados são atribuídos a um ListView que funciona como uma DBGrid em projetos Mobile, como mostra a **Figura 7**.

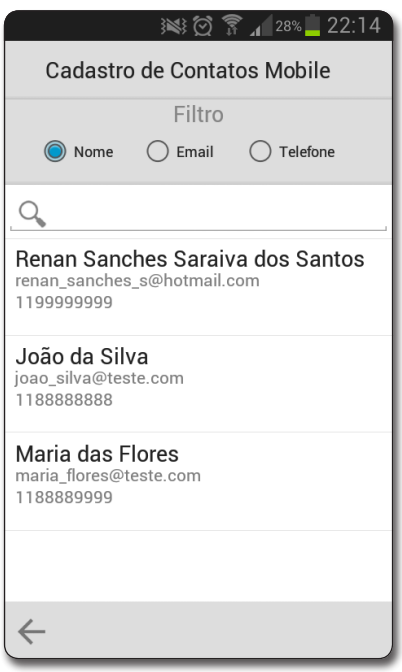

Figura 7. Resultado da Pesquisa de Contatos

Na **Listagem 10** entre as linhas 11 e 13 o formato JSON é "corrigido", removendo os colchetes através do método StringReplace.

Listagem 10. Exibição das empresas cadastradas num listView.

```
01 procedure TFMenu.PesquisaContato(Filtro: Integer);
02 Var
03  item : TListViewItem;
04  Obj: TJSonObject;
05  Par, Par2: TJsonPair;
06  vetor: TJsonArray;
07  i, campo1, campo2 : Integer;
08  resultadoJSON: String;
09 begin
11  resultadoJSON:=idHTTP1.Get(URL+'listaContato');
12  resultadoJSON:=StringReplace(resultadoJSON,'[[','[',[]);
13  resultadoJSON:=StringReplace(resultadoJSON,']]',']',[]);
14
15  Obj := TJSONObject.ParseJSONValue(
        TEncoding.ASCII.GetBytes(resultadoJSON), 0) as TJSONObject;
16  Par := TJSONPair.Create;
17  Par := Obj.Get(0);
18  vetor := TJSONArray.Create;
19  vetor := (Par.JsonValue as TJSONArray);
20  GridContato.Items.Clear;
21  for i := 0 to vetor.Size - 1 do
22  begin
23   Obj := (vetor.Get(i) as TJSONObject);
24   Item := GridContato.Items.Add;
25   case Filtro of
26   0: Begin
27      campo1:=1;
28      campo2:=2;
28    End;
29   1: Begin
30      campo1:=0;
31      campo2:=2;
32    End;
33   2: Begin
34      campo1:=0;
35      campo2:=1;
36    End;
37  end;
38
39  Par2 := Obj.Get(Filtro);
40  Item.Text:=Par2.JsonValue.Value;
41  Par2 := Obj.Get(campo1);
42  Item.Data[TMultiDetailAppearanceNames.Detail1]:=
    Par2.JsonValue.Value;
43  Par2 := Obj.Get(campo2);
   Item.Data[TMultiDetailAppearanceNames.Detail2]:=
    Par2.JsonValue.Value;
44 end;
45 end;
```

A seguir, na linha 15 é realizado o parse, convertendo o valor para um JsonObject. Observe que as linhas 16 a 19 respectivamente preenchem os valores das variáveis Par e Vetor com um JsonPair e um JsonArray respectivamente. Na linha 21 o vetor é percorrido através de um laço for em seu tamanho (quantidade de registros). Por último o ListView é preenchido entre as linhas 39 a 43.

Para finalizar a parte prática, vamos configurar o evento do botão voltar da AbaPesquisa e os eventos OnChange dos RadioButtons, de acordo com a **Listagem 11** para que quando for alterado o filtro, seja realizada novamente a pesquisa jogando o campo selecionado na primeira linha do ListView.

Resumindo, pelos métodos GET e POST do idHTTP podemos trocar mensagens com o servidor REST, que responde sempre em

JSON, onde o resultado pode ser um número/texto ou até mesmo vetor de objetos como é o caso de consultas no banco de dados. Lembrando que não necessariamente o consumidor do método deve ser desenvolvido em Delphi, pois ao utilizar REST+JSON, os serviços podem ser acessados via URL e suas respostas JSON garantem total independência de plataforma, já que JSON assim como XML são linguagens universais para troca de mensagens entre aplicações heterogêneas.

Listagem 11. Rotina para pesquisar novamente ao trocar o filtro.

```
01 procedure TFMenu.RBtnNomeChange(Sender: TObject);
02 var
03  Filtro: Integer;
04  begin
05  if RBtnNome.IsChecked then
06   Filtro:=0
07  Else if RBtnEmail.IsChecked then
08   Filtro:=1
09  Else
10   Filtro:=2;
11  PesquisaContato(Filtro);
12 end;

01 procedure TFMenu.BtnVoltarAbaPesquisaClick(Sender: TObject);
02 begin
03  Abas.ActiveTab:=AbaMenu;
04 end;
```

No cenário em que as empresas se encontram atualmente, os sistemas de informação já estão inseridos e concebidos de forma essencial para a organização, onde a mudança de tecnologia para suportar as exigências citadas acima, é inviável pelo tempo, custo e aprendizado. A forma que se mostra mais viável em todos os sentidos é a integração sistêmica por meio de conceitos relacionados à arquitetura orientada a serviço que oferece uma estrutura de apoio a integração entre plataformas e tecnologias sendo independente de linguagem de programação e sistema.

E através do desenvolvimento de um DataSnap REST pode-se integrar sistemas Desktop, que já rodam nas empresas há tempos, com as novas tecnologias da informação, como é o caso dos dispositivos móveis, aproveitando a estrutura existente na empresa sem a exigência de alto investimento e tempo.

Autor

Renan Sanches Saraiva dos Santos
renan_sacnhes_s@hotmail.com

Programador e analista da March Com. Consultoria e Represent. Trabalha com Delphi desde 2011 utilizando para desenvolvimento de sistemas novos e migração de sistemas legados. Cursando Análise e Desenvolvimento de Sistemas pela FATEC-Taquaritinga.

Links:

Delphi XE5 – Download Trial
https://downloads.embarcadero.com/free/delphi

Delphi XE5 – Página oficial do produto
http://www.embarcadero.com/products/delphi

Datasnap REST Messaging Protocol - Detalhes da invocação de métodos REST
http://docwiki.embarcadero.com/RADStudio/XE5/en/DataSnap_REST_Messaging_Protocol

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

LiveBindings: a evolução no Delphi

Conheça todos os recursos desta tecnologia presente desde o Delphi XE2

O LiveBindings foi lançado juntamente com o FireMonkey (**ver BOX 1**) no Delphi XE2, principalmente porque esta nova plataforma de desenvolvimento não contém os controles DataAware tradicionais como existiam na VCL, então optou-se por criar um mecanismo de Binding semelhante ao que já existe na plataforma .NET, o que obviamente uma ótima ideia da Embarcadero. Vale lembrar que este recurso também foi disponibilizado para os componentes da VCL.

Já era previsto que houvesse melhorias no LiveBindings para as novas versões do Delphi, já que o recurso surgiu no Delphi XE2, onde ainda era um tanto trabalhoso de usar. Em sua primeira versão, as notificações de atualização entre os controles eram feitas manualmente via código. Já nas versões mais recentes do IDE, estas notificações são automáticas, bastando o desenvolvedor ligar os controles de forma gráfica e intuitiva.

Foram criados dois facilitadores para esta tarefa que são o LiveBindings Designer e o LiveBindings Wizard.

DataBinding

O DataBinding é uma técnica de ligação entre duas fontes de dados que nos permite manter a sincronização destes, onde uma alteração feita por um elemento reflete automaticamente nos elementos ligados a ele. Para isto temos um objeto de origem (source) e um ou mais objetos de destino (target). Alterações no objeto de origem irão refletir automaticamente nos objetos de destino.

O uso mais comum de DataBinding é o de ligar campos do banco de dados ou atributos de uma classe a um controle de tela, como Edits, Memos, Grids, etc.

Podemos ter três tipos de ligações entre estes elementos:

- OneWay: onde somente o objeto de origem atualiza o de destino;
- TwoWay: neste os dois fazem atualizações no outro;
- OneWayToSource: aqui o destino quem faz atualizações no objeto de origem.

Fique por Dentro

Através de exemplos comparativos entre as versões do Delphi, é mostrado como o recurso de Live Bindings amadureceu. Os exemplos mostram como a praticidade se tornou o foco em sua utilização, colocando lado a lado situações que sofreram melhoras, por exemplo, a notificação de mudanças nos bindings, o surgimento de um assistente para criação de Bindings, binding de objetos não visuais e outras.

BOX 1. FireMonkey

O FireMonkey é o novo Framework de desenvolvimento de aplicações criado pela Embarcadero a partir do Delphi XE2. Seu propósito surgiu através da necessidade de criar aplicações com visual mais rico e que sejam multiplataforma. Os recursos gráficos do FireMonkey fazem uso da GPU do hardware, possibilitando o trabalho com aplicações HD, 3D. A versão FireMonkey 2, presente no Delphi XE3 trouxe algumas novidades como Actions, Anchors, Touch e Gesture, recursos estes que já existiam na VCL.

LiveBindings

O LiveBinding é um mecanismo de DataBinding que foi incorporado na versão XE2 do Delphi e que tem o objetivo de fazer a ligação de fontes distintas de dados através de expressões. Estas expressões são relacionais, conhecidas como Binding Expressions, podendo ser unidirecional ou bidirecional, que define o sentido do fluxo de dados na ligação.

- Unidirecional: a alteração é feita apenas na origem, ou seja, a propriedade de destino apenas reflete os dados da origem.
- Bidirecional: a alteração é feita na origem e no destino, ou seja, qualquer alteração é refletida tanto na origem quanto no destino.

Por meio destas expressões podemos ter qualquer objeto ligando com qualquer outro, simplesmente criando esta expressão de ligação entre eles.

No LiveBinding temos os objetos que são os Source Objects (objetos de origem) e os Control Objects (objetos de controle).

Há a possibilidade de um mesmo objeto ser tanto de origem quanto de controle. Os componentes disponíveis podem ser vistos na **Figura 1**.

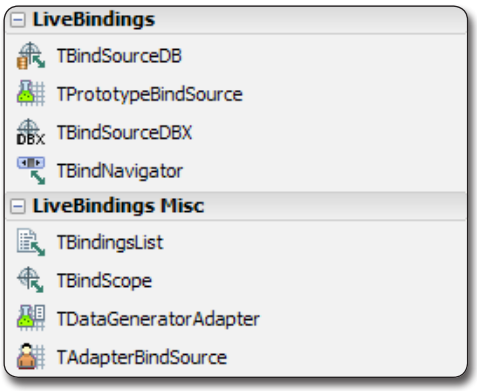

Figura 1. Componentes LiveBindings no Delphi XE4

LiveBindings Designer

Este editor pode ser visualizado usando o menu View/LiveBindings Designer, ele nos exibe os componentes que temos em nossa tela e a partir de agora podemos fazer a ligações deste componentes simplesmente traçando uma linha simples entre as propriedades dos objetos.

A partir desta versão, o próprio Delphi se encarrega de disparar as notificações entre os componentes, não precisamos mais programar nenhum método nem codificar nenhum evento para esta tarefa, como tínhamos anteriormente no Delphi XE2.

A seguir uma breve descrição de cada botão do editor conforme a ordem da **Figura 2**.

- Fit in Window: redimensiona os itens para se adequar ao tamanho atual do editor;
- Zoom In: aumenta o tamanho dos itens do editor;
- Zoom Out: diminui o tamanho dos itens do editor;
- Restore Actual Size: restaura o tamanho padrão dos itens;
- LiveBindings Wizard: abre um wizard que ajuda na criação de Bindings.

Figura 2. Botões do Editor

No Editor de Bindings também temos os chamados Layers ou camadas, que são diferentes formas de visualização que podemos ter das ligações entre os componentes, como pode ser visto na **Figura 3**. A ideia é dar foco a alguns componentes e ligações que desejamos. Isto é muito útil quando há muitas ligações dentro do formulário, como por exemplo dois ClientDataSets ligado a vários controles. Neste caso poderíamos ter duas visualizações, uma para cada ClientDataSet.

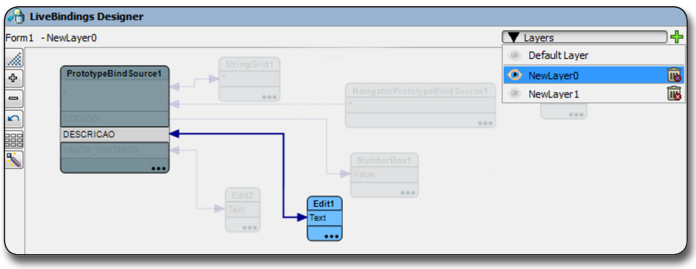

Figura 3. Layers(camadas) do Wizard

Existem alguns componentes, como rótulos, que não temos a necessidade de termos no nosso editor de Bindings e que acabam poluindo a tela. Podemos deixar invisíveis estes controles para o editor, bastando apenas acionarmos o botão direito dentro do editor e escolher a opção "Show/Hide elements...". Será aberta uma janela com todos os itens do formulário, onde basta desmarcar os controles que não desejamos que sejam exibidos no editor, conforme a **Figura 4**.

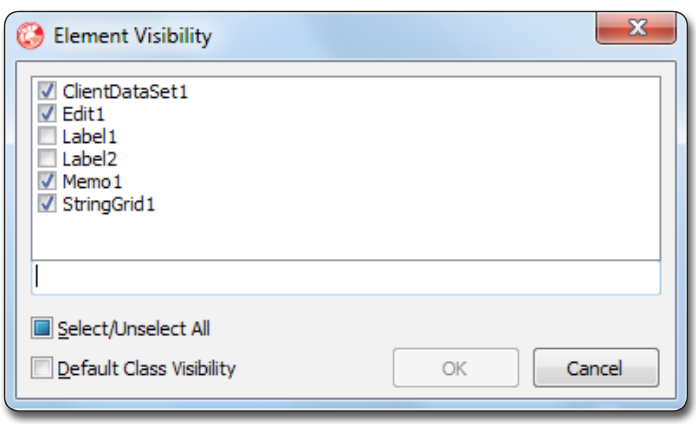

Figura 4. Ocultando elementos do editor

Olhando para o editor vemos que as principais propriedades de cada componente ficam disponíveis para fazermos a ligação, porém, podemos adicionar as propriedades que estão ocultas de cada componente, basta clicarmos nos "três pontos" no canto inferior direito de cada componente do editor que ele vai trazer todas as propriedades disponíveis naquele componente, e então podemos escolher quais desejamos que estejam disponíveis no editor, conforme a **Figura 5**.

Acionando o botão direito dentro do editor, temos outro interessante recurso, que é o de exportar nosso diagrama como imagem, conforme mostra a **Figura 6**. Podemos configurar várias opções relacionadas ao tamanho do diagrama. Temos um preview do resultado e podemos salvar em vários formatos de imagens. Isso é interessante para documentação da aplicação que estamos desenvolvendo.

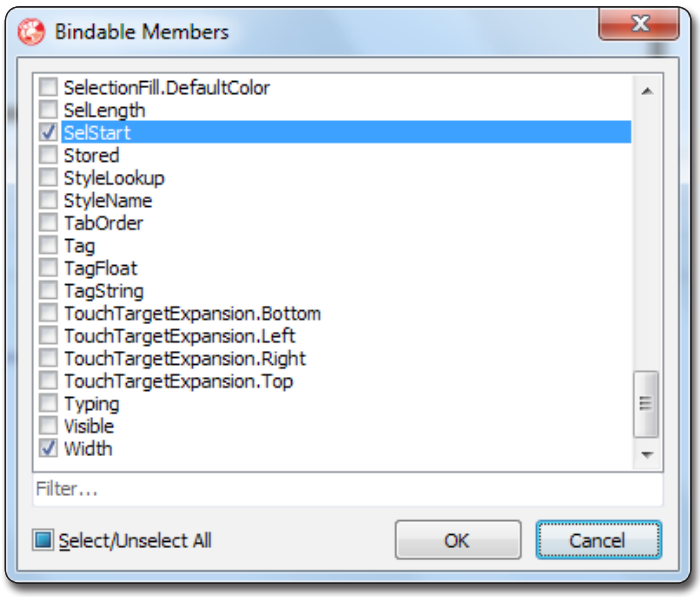

Figura 5. Propriedades ativas ou inativas no Editor

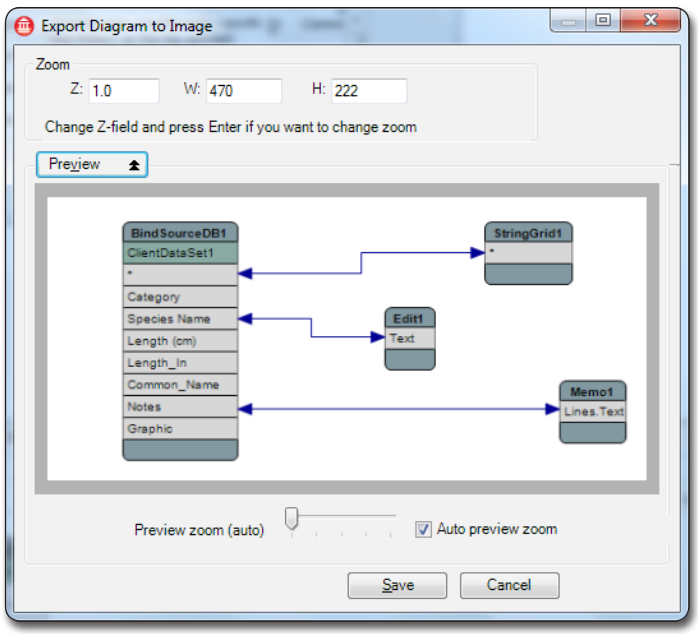

Figura 6. Exportando diagrama para imagem

Notificações no Delphi XE2

Na versão XE2 do Delphi, para refletirmos o texto de um Edit no Caption de um Button tínhamos que fazer as seguintes operações. Ir na propriedade LiveBindings e escolher New LiveBinding.

Após escolher a opção TBindExpression, era criado o componente TBindingsList onde deveríamos configurar na nossa expressão as seguintes opções:

- Control Expression: propriedade do Button afetada, no caso Text.
- Source Component: componente de origem, no caso o Edit.
- Source Expression: expressão, que pode ser simplesmente Text ou qualquer função adicional.

Para o correto funcionamento do exemplo ainda precisava-se implementar um procedimento de notificação na seção published da unit, para que a cada troca de dados no Edit os controles ligados sejam notificados, conforme a **Listagem 1**.

Listagem 1. Evento de notificação

```
01 procedure Notify(Sender: TObject);
02 begin
03   BindingsList1.Notify(Sender, '');
04 end;
```

Finalmente, no evento OnChange do Edit devíamos vincular a procedure Notify e ao executar tudo funcionava conforme o planejado.

Note que para apenas uma relação tivemos uma série de configurações a fazer além da codificação, que não deveria ser necessária, já que isso faz parte do recurso de LiveBindings e não é uma lógica de nossa aplicação. Veremos a seguir o quanto isso ficou fácil nas versões recentes do IDE.

Notificações no Delphi XE4

Para termos o mesmo exemplo no Delphi XE4 basta segurarmos o botão esquerdo do mouse da propriedade Text do Edit até a propriedade Text do Button. Pronto, basta executarmos a aplicação e temos o nosso exemplo em funcionamento, conforme a **Figura 7**.

É claro que para o tratamento avançado desta ligação temos que ir na expressão do componente TBindingsList, mas sem dúvida este Wizard nos traz muita produtividade no desenvolvimentos de telas com vários controles interligados.

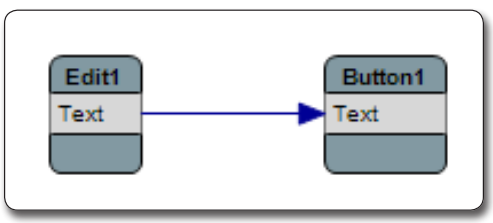

Figura 7. Ligação visual dos componentes

LiveBindings Wizard

Temos disponíveis a partir do Delphi XE3 também o LiveBindings Wizard, onde é possível acessá-lo acionando o botão direito do mouse em qualquer componente da tela e escolher a opção referida.

Será aberta uma tela para ajudar o desenvolvedor a criar a ligação entre os objetos. Essa Wizard facilita muito o desenvolvedor a criar as ligações. Vale lembrar que este é muito sensível ao contexto, portanto, dependendo do componente selecionado, mudam as opções disponíveis no Wizard.

Esse Wizard é dividido em duas partes, no lado esquerdo ficam os passos envolvidos no assistente e no lado direito as opções disponíveis ao desenvolvedor naquele momento.

Temos várias opções de ligações no assistente, como as seguintes:

- Ligar um controle a um campo Field;
- Ligar um grid a um DataSource;
- Ligar a propriedade de um componente à propriedade de outro componente;

Nota

Para acessarmos o Wizard do LiveBindings no Delphi XE3 pelo botão direito do mouse devemos mudar uma configuração do IDE, pois este recurso não vem habilitado por padrão. Para habilitá-lo basta acessar Tools>Options>LiveBindings e marcar a opção "Display LiveBindings Wizard in rigth-click menu".

LiveBindings, ClientDataSet e Action Lists

O componente TActionList foi disponibilizado no FireMonkey a partir da versão XE3 e junto a ele vieram algumas Actions padrões do LiveBindings, conforme podemos notar na **Figura 8**. Estas ações são semelhantes aquelas que encontramos no componente TBindNavigator e estão relacionadas às funções do ClientDataSet.

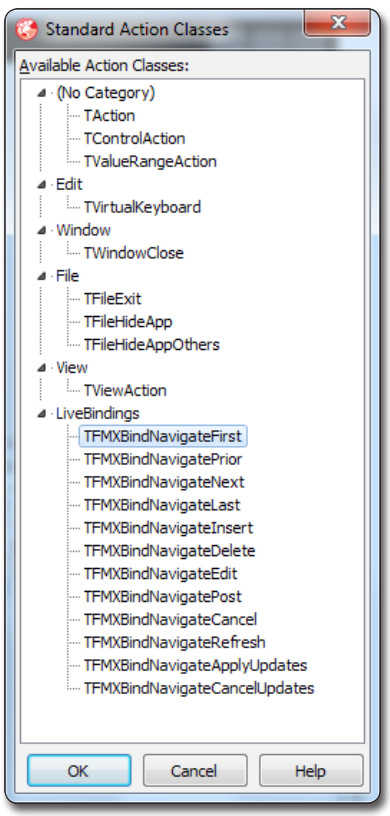

Figura 8. Actions relacionadas à LiveBindings disponíveis

Agora vejamos a facilidade de utilização dos componentes do FireMonkey em conjunto com o LiveBindings e que são necessárias muito poucas linhas de código para fazer a ligação entre os componentes.

Primeiro criamos um novo projeto FireMonkey e adicionamos o componente TClientDataSet, ligando o mesmo a qualquer origem de dados (XML, CDS, SqlDataSet, etc), em seguida, adiciona-se o componente TGrid na tela, acionando sobre ele o botão direito do mouse>LiveBindings Wizard. Então temos a nova janela de Wizard e a opção de conectar o Grid a um DataSource. Na guia seguinte optamos por fazer o link ao componente ClientDataSet adicionado anteriormente. Podemos ainda adicionar um DataSourceNavigator e finalizar o Wizard.

Visualizando o formulário há agora novos componentes. O TBindNavigator para navegar/incluir/alterar/excluir registros, note também que foi criado um TBindDataSource que tem uma função muito semelhante ao DataSource presente na VCL, inclusive ele possui internamente um DataSource e um DataSet para fazer essa junção.

Outro componente gerado é o TBindingList que é o componente que liga a nossa grade de dados ao BindSourceDB. Ele cria internamente o objeto TLinkGridToDataSource onde fica armazenado o grid de dados e o DataSource de origem das informações. Através da propriedade Columns podemos ainda definir quais colunas queremos que sejam exibidas. Agora adicione um TActionList e uma ação TFMXBindNavigateNext através do Action List Editor e aponte a propriedade DataSource para o nosso BindDataSource já presente na tela. Para finalizar atribuímos a propriedade Action do botão na ação que acabou de ser criada. Executamos a aplicação e vemos que não foi necessária nenhuma linha de código para fazer toda essa combinação. Podemos usar o botão Next para avançarmos no grid. Desta mesma forma poderíamos ter usado qualquer outra operação do ClientDataSet, como Insert, Edit, Delete e etc.

É possível ainda adicionar um Edit e um Label vinculado ao campo Description. Basta acionarmos o LiveBindings Wizard pelo ClientDataSet, escolhermos a opção "Link a field to ClientDataSet to a control", escolher o campo Description e em seguida selecionar o controle que desejamos, no caso TEdit. No último passo marcamos a opção para adicionar um Label. O resultado é o mostrado na **Figura 9**.

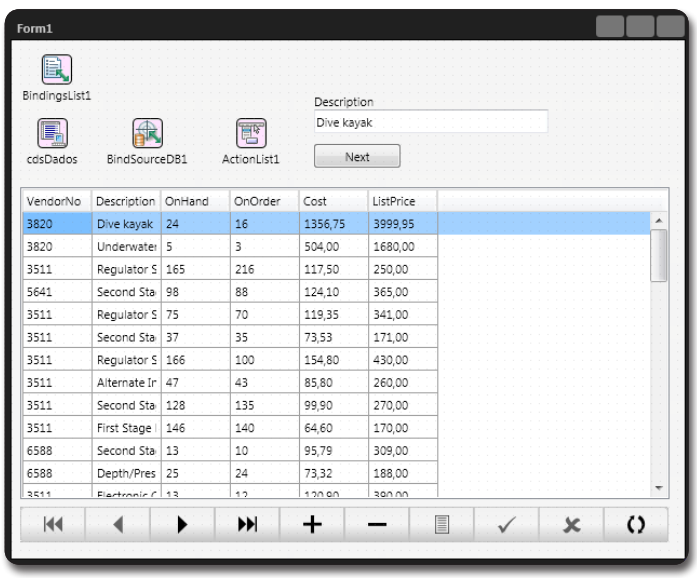

Figura 9. Componentes para ligação do Grid

Observe agora que foi criada mais uma expressão no nosso TBindingsList, o tipo é TLinkControlToField (**Figura 10**) e como o nome diz serve para ligarmos um controle de tela a um TField do ClientDataSet.

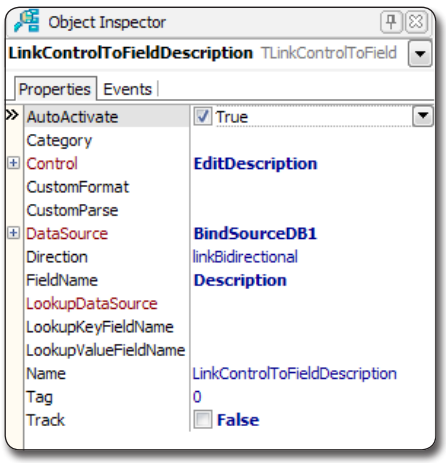

Figura 10. Expressão de ligação entre TField e Control

Criação de Protótipos

Na versão XE3 do Delphi também tivemos a adição de um novo componente chamado TPrototypeBindSource. Sua função é criar protótipos de base de dados para testes. Agora não somos mais obrigados a criarmos um banco de dados para testar nossa aplicação, nem ligarmos um ClientDataSet a um XML, basta utilizarmos este componente. É extremamente útil para criarmos aplicações de modelo.

Para usá-lo, basta clicar duas vezes e acionar Add New. Será apresentada uma janela conforme a **Figura 11** onde informamos o nome do campo e tipo do campo. Veja que ele é semelhante à configuração dos TFields do ClientDataSet, porém, não é persistido em nenhum lugar.

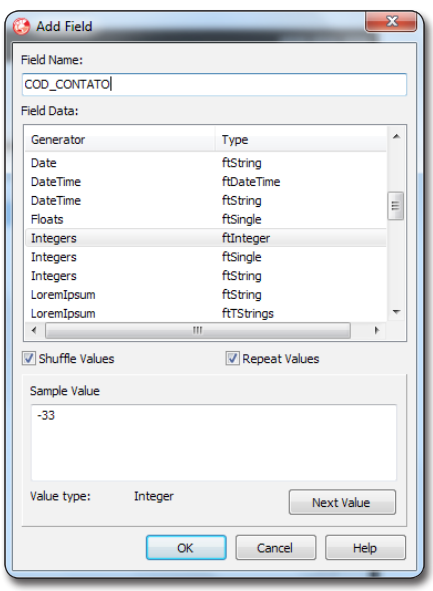

Figura 11. Adicionando campos no Protótipo

Cada um dos campos adicionados pode estar ligado a um Generator que tem a função de gerar dados de teste para o campo, facilitando bastante o desenvolvimento de protótipos a aplicativos de demonstração. Agora podemos adicionar Grids, Edits, Labels e através do Live-Binding Wizard, vincularmos aos campos, conforme podemos observar na **Figura 12**.

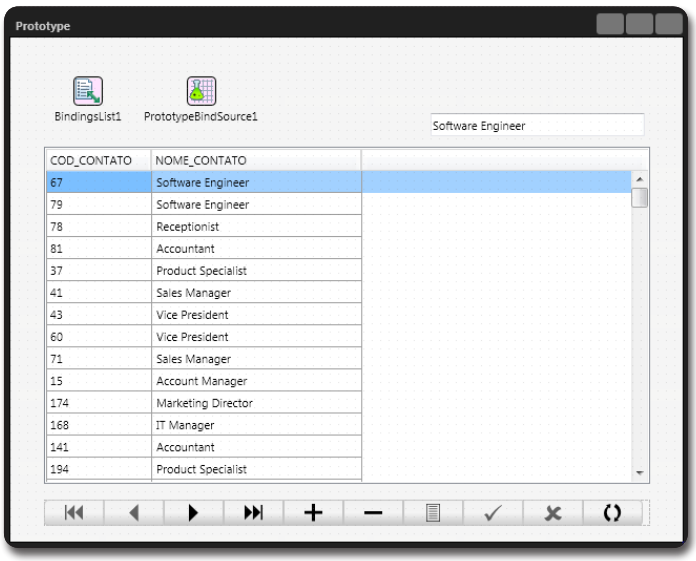

Figura 12. Demonstração do protótipo final

Criando nosso próprio Generator

Podemos desenvolver nosso próprio gerador de dados. Nesse caso simularemos um gerador de IDs em sequência. Para isso, devemos criar um novo pacote para o gerador, além disso, implementaremos uma classe chamada TIDsGenerator que herdará da classe TDelegateValueGenerator, classe base para todos os geradores e que possui um método virtual CreateDelegate que precisamos sobrescrever para implementarmos o gerador personalizado. O código da classe é exibido na **Listagem 2**.

Para retornarmos os dados usamos a classe TTypedListValueGeneratorDelegate (linha 27) que através de Generics pode receber qualquer classe. Observe que o segundo parâmetro é um TArray de Integer, porque definimos que o tipo retornado seria Integer. Se tivéssemos informado String antes de chamar o construtor Create, seria retornado um Array de Strings, se passado um Double seria retornado um Array de Doubles e assim por diante. Aqui podemos perceber os Generics sendo utilizados para resolver este problema da compatibilidade de tipos (Type Casting), ou seja, pode-se criar uma lista de qualquer classe ou tipo primitivo para retornarmos.

Esta classe deve ser incluída em um pacote e, após compila-lo, deve-se realizar sua instalação no IDE para que ele apareça na lista de geradores do componente TPrototypeBindSource, conforme **Figura 13**.

Na **Figura 14** foi adicionado a um formulário um componente grid e vinculado ao gerador com os IDs. Sua ligação foi feita pelo gerador ContactName, padrão do Delphi.

Listagem 2. Classe TIdsGenerator01 unit uIDsGenerator;

```
02
03 interface
04
05 uses
06  Data.Bind.ObjectScope, System.Generics.Collections;
07
08 type
09  TIdsGenerator = class(TDelegateValueGenerator)
10  protected
11   function CreateDelegate: TValueGeneratorDelegate; override;
12  end;
13
14 implementation
15
16 { TDataGenerator }
17
18 function TIdsGenerator.CreateDelegate: TValueGeneratorDelegate;
19 var
20  ListaIds: TList<Integer>;
21  i: Integer;
22 begin
23  ListaIds := TList<Integer>.Create;
24  try
25   for i := 0 to 50 do
26    ListaIds.Add(i);
27   Result := TTypedListValueGeneratorDelegate<Integer>.Create(
          Options, ListaIds.ToArray);
28  finally
29   ListaIds.Free;
30  end;
31 end;
32 initialization
33  RegisterValueGenerator('Gerador IDs', [ftInteger],
34   TValueGeneratorDescription.Create(TIdsGenerator, '',
     'uIDsGenerator'));
35 finalization
36  UnRegisterValueGenerator('Gerador IDs', [ftInteger], '');
37 end.
```

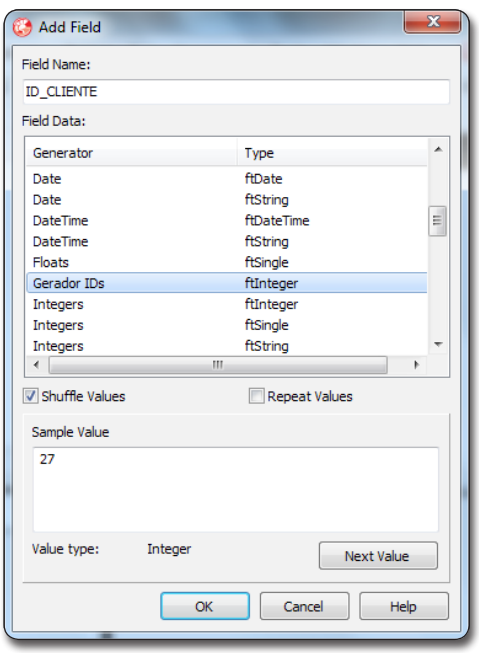

Figura 13. Gerador de IDs

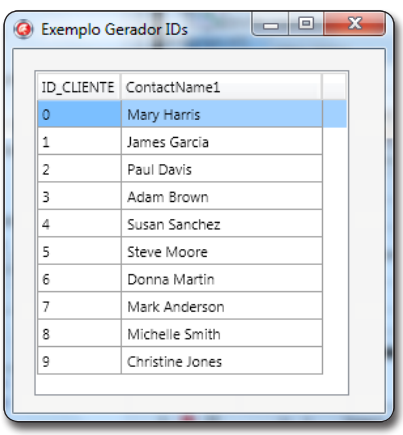

Figura 14. Gerador em execução

Investindo um tempo para criação de geradores personalizados fica muito mais fácil a criação de protótipos de aplicação. Poderíamos gerar Datas de Nascimentos aleatórias, e-mails, CPFs, existem muitos dados de testes que os desenvolvedores utilizam no dia a dia. Seria interessante ainda termos um pacote com todos os nossos geradores personalizados e podermos utilizar em qualquer aplicação que criemos, lembrando que podemos usar este recurso tanto em aplicações VCL quanto FireMonkey, o que torna isso ainda mais vantajoso.

Componente TBindingsList

Este componente é o grande responsável por fazer a ligação entre dois objetos. É nele que fica a expressão de ligação. Se colocarmos um Edit e um Button e realizarmos uma ligação através do Editor das propriedades Text dos dois componentes, automaticamente será adicionado um componente do tipo TBindingsList com um item internamente, conforme mostra a **Figura 15**.

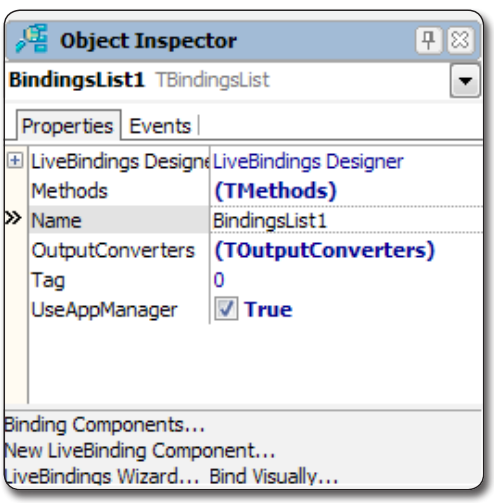

Figura 15. Propriedades do TBindingsList

As propriedades mais interessantes de analisarmos neste componente são a lista de métodos e formas de conversão de dados, através das propriedades Methods e OutputConverters que nos mostram quais são os métodos e conversões de dados que podemos usar nas propriedades dos objetos que ligamos.

Se atentarmos aos itens do TBindingsList, veremos que foi criado um link entre os dois componentes. Deste modo, podemos olhar com atenção suas propriedades e entendermos como funciona o mecanismo de Binding do Delphi, conforme mostrado na **Figura 16** e na **Tabela 1,** respectivamente.

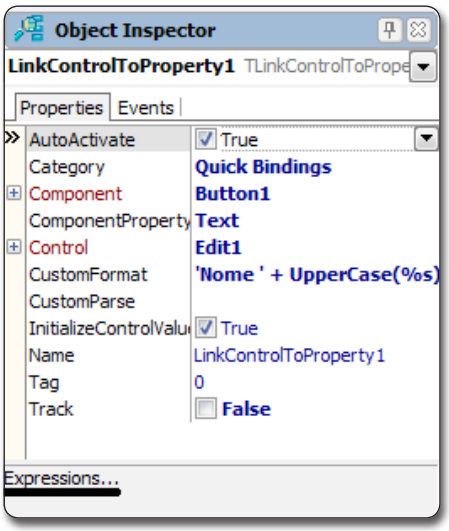

Figura 16. Propriedades do Link criado

Propriedade	Função
Category	Categoria da Ligação, semelhantes às categorias existentes no ActionList
Component	Componente de destino que receberá os dados
ComponentProperty	Propriedade do componente de destino que receberá os dados
Control	Origem dos dados
CustomFormat	Formatações que desejamos no objeto de destino
CustomParse	Conversões de dados para o objeto de destino

Tabela 1. Propriedades dos componentes Link

Observe que aqui vimos somente um tipo de Binding que é o TLinkControlToProperty, mas existem muitos outros, como pode ser visto na **Figura 17**. Cada um tem uma função específica e que se encaixa em determinadas situações. Devemos evitar o uso dos DB Links, pois foram marcados como obsoletos e só permanecem por questão de retro compatibilidade.

Um evento importante contido em todas as expressões é o OnEvalError. Esta recebe como parâmetro um objeto do tipo Exception e intercepta qualquer tipo de exceção ocorrida em nossa expressão, exibindo-a de forma familiar ao usuário, como na **Listagem 3**.

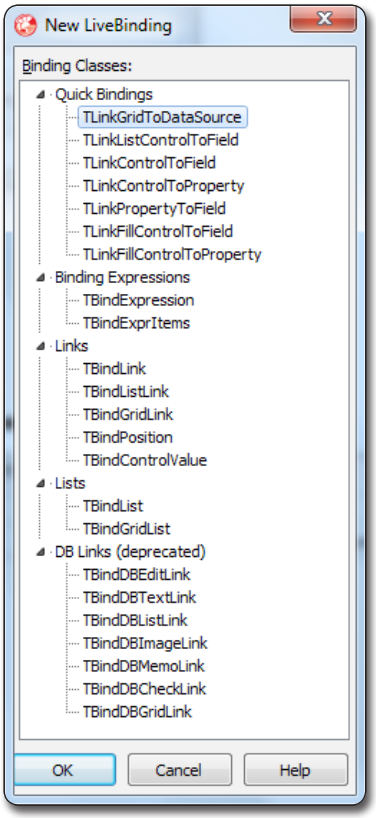

Figura 17. Tipos de Bindings disponíveis

Listagem 3. Manipulando Exceções nas expressões

```
01 procedure TForm1.LinkControlToPropertyTextEvalError(Sender: TObject;
     AException: Exception);
02 begin
03   raise Exception.Create('Classe do Erro:' + AException.ClassName +
       #13#10 + 'Mensagem:' + AException.Message);
04 end;
```

Ligando componentes não visuais no Delphi XE2

Neste exemplo veremos como tínhamos que fazer a ligação entre objetos de negócio e componentes visuais no Delphi XE2. A ideia é ligar um simples TEdit a uma propriedade de uma classe não visual.

A interface é muito simples e pode ser visualizada na **Figura 18**. O código pode ser visualizado na **Listagem 4**.

Na linha 5 temos a declaração das duas units necessárias para podermos implementar este exemplo. Criamos a classe TPessoa com uma única propriedade: "Nome", optou-se por criar na mesma unit do formulário para facilitar (linhas 06 a 12).

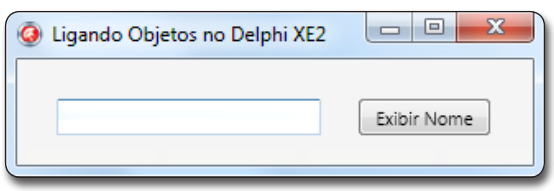

Figura 18. Interface do exemplo

Listagem 4. Código completo da Unit

```
01 unit ufrMain;

02 interface

03 uses
04  FMX.Types, FMX.Controls, FMX.Forms, FMX.Dialogs, FMX.Edit,
    System.Classes,
05  System.Bindings.Expression, System.Bindings.Helper;

06 type
07  TPessoa = class
08  strict private
09   FNome: string;
10  public
11   property Nome: string read FNome write FNome;
12  end;

13  TfrmMain = class(TForm)
14   Edit1: TEdit;
15   Button1: TButton;
16   procedure btnExibirClick(Sender: TObject);
17   procedure FormDestroy(Sender: TObject);
18   procedure edtNomeChange(Sender: TObject);
19   procedure FormCreate(Sender: TObject);
20  private
21   FPessoa: TPessoa;
22  public
23  end;

24 var
25  frmMain: TfrmMain;
26  BindingExpression: TBindingExpression;

27 implementation
28 {$R *.fmx}

29 procedure TfrmMain.FormCreate(Sender: TObject);
30 begin
31  FPessoa := TPessoa.Create;
32  BindingExpression := TBindings.CreateManagedBinding(
33   [TBindings.CreateAssociationScope([Associate(Edit1, 'Edit')])],
34   'Edit.Text',
35   [TBindings.CreateAssociationScope([Associate(FPessoa, 'Pessoa')])],
36   'Pessoa.Nome', nil);
37 end;

38 procedure TfrmMain.edtNomeChange(Sender: TObject);
39 begin
40  TBindings.Notify(Sender, 'Text');
41 end;

42 procedure TfrmMain.btnExibirClick(Sender: TObject);
43 begin
44  ShowMessage(FPessoa.Nome);
45 end;

46 procedure TfrmMain.FormDestroy(Sender: TObject);
47 begin
48  FPessoa.Free;
49 end;

50 end.
```

Na linha 26 é definida uma variável de escopo global do tipo TBindingExpression, esta será responsável por armazenar a expressão que realizará a ligação dos dados.

Entre as linhas 29 a 37 há a codificação do evento OnCreate do formulário. Primeiramente é instanciada a classe TPessoa (linha 31). Na linha 32 é realizado o mecanismo de Binding entre TPessoa.Nome e Edit.Text. Para realizar essa associação, é utilizado o método CreateManagedBinding da classe TBindings. O primeiro parâmetro é um array de objetos de Origem dos dados. Já o segundo é a expressão de ligação que será utilizada para fazer o Binding. O terceiro parâmetro é um Array de objetos de destino, que receberá os dados do LiveBinding, já o quarto parâmetro é a expressão que será utilizada para armazenar o valor de saída. A função Associate serve para vincularmos um objeto a uma String, já que o LiveBinding não trabalha diretamente com os objetos e sim com expressões, por isso a necessidade desta associação.

Somente com esta ligação feita no evento OnCreate nosso exemplo ainda não funcionaria. Isso porque na versão anterior do Delphi não havia o recurso de auto notificação, ou seja, o framework não gerenciava os objetos relacionados para a atualização automática. Precisamos então notificar manualmente, ou seja, devemos codificar o evento OnChange do Edit e notificar o componente da mudança (linha 40).

No evento OnClick do botão simplesmente é exibido o conteúdo da propriedade Nome do objeto FPessoa. No evento OnDestroy objeto FPessoa é liberado da memória.

Como se percebe neste exemplo tudo funcionou, no entanto, tivemos bastante trabalho para fazer as ligações entre os componentes visuais e os objetos. Isso melhorou muito nas novas versões do Delphi como veremos a seguir.

Ligando componentes não visuais no Delphi XE3 e XE4

A partir do Delphi XE3 foram adicionados novos recursos que vieram nos trazer facilidades para trabalhar com Bindings de componentes não visuais.

Primeiramente, vamos criar um novo projeto FireMonkey, e em seguida, implementar uma nova classe chamada TContato, conforme ilustra a **Listagem 5**. É uma classe muito simples que dispõe de um construtor personalizado para facilitar a população dos Fields. Entre as linhas 11 e 13 definimos as propriedades públicas da nossa classe TContato. Observe que na linha 10 foi definido o construtor. Este foi implementado a partir da linha 16, fazendo com que os atributos da classe recebam os valores diretamente em sua criação.

Finalizada a implementação, o próximo passo é definir os campos da classe anteriormente criada, no componente TDataGeneratorAdapter, através da propriedade FieldDefs. Basta adicioná-los de acordo com o tipo desejado (**Figura 19**).

Agora adicionamos o componente TAdapterBindSource e ligamos nele o TDataGeneratorAdapter através da propriedade Adapter. Em seguida ligamos nossos controles de tela (edits) ao nosso Adapter.

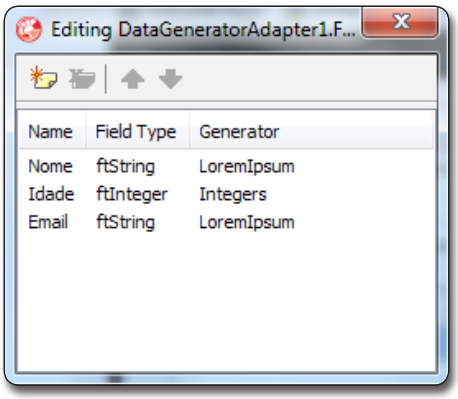

Figura 19. Criando os Fields de acordo com a Classe

Para finalizar a ligação entre nossa classe e o componente TAdapterBindSource, no evento OnCreateAdapter escrevemos o código da **Listagem 6**.

Na linha 4 simplesmente é instanciado o objeto da classe TContato atribuindo seus Fields. Na linha 5 usamos a variável ABindSourceAdapter (do parâmetro) e define-se o vínculo entre o TAdapterBindSource e a classe TContato através da nova instância de TObjectBindSourceAdapter. Observe que é usado Generics (ver **BOX 2**) para dizer qual o tipo (TContato) e o objeto (FContato) que deve ser usado para fazer a inicialização. Esta classe pode ser usada para fazer vínculo de qualquer objeto a qualquer TAdapterBindSource.

Listagem 5. Classe TFuncionario

```
01 unit uContato;

02 interface

03 type
04  TContato = class
05  private
06    FNome: string;
07    FIdade: Integer;
08    FEmail: string;
09  public
10    constructor Create(Nome: string; Idade: Integer; Salario: Double);
11    property Nome: string read FNome write FNome;
12    property Idade: Integer read FIdade write FIdade;
13    property Email: string read FEmail write FEmail;
14  end;

15 implementation

16 constructor TFuncionario.Create(Nome: string; Idade: Integer; Email:
   string);
17 begin
18  FNome := Nome;
19  FIdade := Idade;
20  FEmail := Email;
21 end;

22 end.
```

Listagem 6. Evento OnCreateAdapter Objeto

```
01 procedure TForm2.AdapterBindSource1CreateAdapter(Sender: TObject;
02   var ABindSourceAdapter: TBindSourceAdapter);
03 begin
04  FContato := TContato.Create('Filipe', 25,
     'filipe.dalepiane@gmail.com');
05  ABindSourceAdapter := TObjectBindSourceAdapter<TContato>.Create
    (Self, FContato);
06 end;
```

BOX 2. Generics

Generics é um recurso presente desde o Delphi 2009 que permite a criação de estruturas genéricas. Pode-se definir o tipo de uma lista de objetos que serão adicionados. Se o objeto adicionado não for o especificado, teremos um erro ainda em tempo de compilação, evitando assim erros de TypeCasting que ocorriam em versões anteriores do Delphi.

Você pode estar se perguntando no momento, e se eu quiser um grid com uma lista de objetos de TContato, sem problemas, basta adicionar outro TAdapterBindSource a através do próprio Wizard, ligá-lo com o componente Grid. Em seguida, basta configurar novamente o evento OnCreateAdapter como sugere a **Listagem 7.**

Listagem 7. Evento OnCreateAdapter Lista

```
01 procedure TForm2.AdapterBindSource2CreateAdapter(Sender: TObject;
     var ABindSourceAdapter: TBindSourceAdapter);
02 begin
03  FListaContatos := TObjectList<TContato>.Create;
04  FListaContatos.Add(TContato.Create('Filipe', 25, filipe@gmail.com'));
05  FListaContatos.Add(TContato.Create('Cíntia', 27, 'cintia@gmail.com'));
06  FListaContatos.Add(TContato.Create('Julio', 32, 'julio@gmail.com'));
07  ABindSourceAdapter := TListBindSourceAdapter<TContato>.Create(Self,
     FListaContatos);
08 end;
```

Observe que a única diferença entre as duas listagens é que na primeira é passado um único objeto, enquanto na segunda, cria-se uma lista de objetos de TContato através da classe TObjectList, presente na unit Generics.Collections.

O diagrama da **Figura 20** mostra como ficaram as ligações finais da aplicação.

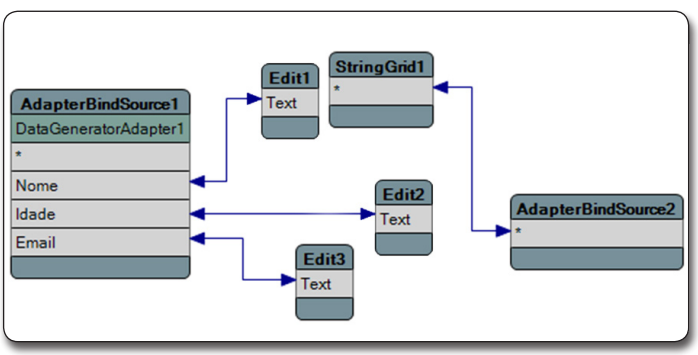

Figura 20. Diagrama final de ligações entre os componentes do formulário

Sistema de créditos DevMedia

Conteúdo **completo** ao seu alcance

Com o sistema de créditos DevMedia você pode comprar individualmente mais de 4.000 vídeo aulas, 8.000 artigos, além de palestras e vídeos com formações completas.

TODAS as vídeo aulas que você sempre quis ver a partir de

R$ 1,20

ECONOMIA: você não precisa assinar 8 produtos diferentes para ter acesso ao conteúdo completo do site!

Saiba mais sobre o Sistema de Créditos!

www.devmedia.com.br/creditos

DEVMEDIA

Ao executar a aplicação, é possível verificar o funcionamento tanto de um objeto quanto de uma lista de objetos. A **Figura 21** ilustra o resultado final.

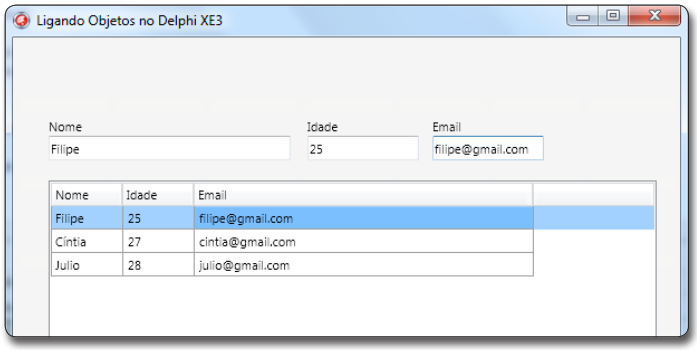

Figura 21. Aplicação em funcionamento

Novo componente TParamsAdapter do Delphi XE4

A mais nova versão do Delphi trouxe consigo um novo componente na paleta LiveBindings, o TParamsAdapter. Este componente surgiu para podermos ligar componentes de acesso a dados do DBExpress ao componente TAdapterBindSource.

Como já abordado, o TAdapterBindSource foi criado para realizar o Binding de qualquer fonte de dados através de um Adapter.

Anteriormente, na ligação de classes utilizamos o TDataGeneratorAdapter para definir o Binding, já no TParamsAdapter pode-se vincular qualquer SQLDataSet, SQLQuery, SQLServerMethod e/ou SQLStoreProc. Após isso, basta liga-lo ao TAdapterBindSource através da propriedade Adapter e trabalhar normalmente, manipulando o evento OnCreate como feito anteriormente.

Como vimos, há vários tipos de Binding que podem ser usados em nossas aplicações, ligações em componentes, componentes em DataSets, componentes a Objetos e muitos outros tipos. Pode-se ainda estender o framework de modo a implementar o padrão MVC por exemplo.

O FireMonkey veio para ficar e se quisermos que nossas aplicações tenham interfaces ricas e multiplataforma, devemos dominar o recurso de LiveBinding para desenvolvermos aplicações de qualidade, seguras e ágeis.

Autor

Filipe Dalepiane
Analista Desenvolvedor Delphi a 5 anos, utilizando arquitetura multicamadas, com os Banco de Dados Firebird e DB2. Colunista da revista Clube Delphi. Adepto de boas práticas de programação, modelagem UML, Orientação a Objetos e Design Patterns. Graduando Ciência da Computação pela UNIJUÍ.

Links:

Data Binding Overview
http://msdn.microsoft.com/en-us/library/ms752347.aspx

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DEVMEDIA

WPTools: edição, correção ortográfica e mala direta de textos no Delphi XE 3

Criação de um editor completo com a WPTools

WPTools é uma suíte de componentes pagos especializados para o processamento de texto que está no mercado desde 1996. Sua instalação é bem simples, consiste de um instalador que descompacta os arquivos necessários para uma pasta de sua escolha. A última versão da WPTools oferece suporte para as versão XE à XE5. Veja na seção **Links** como realizar o download da versão trial.

Uma vez descompactado o conteúdo, basta abrir e compilar os pacotes (*.dpk) localizados na pasta WPTools\Dxxx (onde xxx é sua versão de Delphi) e então clicar em install. Depois é preciso inserir no Library Path do Delphi, acessando Tools>Options>Delphi Options>Library, a localização da pasta das units do componente (WPTools\Dxxx, WPTools\Source).

Características

O principal diferencial técnico está no fato de que a WPTolls é desenvolvida 100% em Delphi e seu motor de processamento de texto não utiliza a DLL Richedit do Windows ou qualquer outra DLL de terceiros. Possui um editor WYSIWYG que oferece visualização de layouts, cabeçalhos, rodapés, tabulação, estilos de parágrafo e oferece suporte total ao UNICODE.

Também oferece recursos especiais como a divisão de tela para edição, expansão/recolhimento de parágrafos e exibição interativa de documentos. A realização de mesclagem de informações vindas de qualquer fonte de dados, também conhecida como mala direta, e permite a utilização de textos sem formatação, com formatação e também imagens.

Além de texto RTF o editor também permite a edição de HTML e XML, controle da área de transferência com restrições de segurança, por exemplo, um texto copiado para a memória não sai do editor.

Um editor básico

A WPTools é bem completa e oferece controles visuais prontos para o uso. Para demonstrá-los vamos criar um novo projeto no Delphi XE 3 (File>New>VCL Forms Application). O formulário principal tem seu nome modificado para EditorBasicoF e ao salvar o projeto salvamos como EditorBasico.

Fique por Dentro

Este artigo vem apresentar uma solução para os requisitos de edição de texto e criação de malas diretas que fazem parte de muitos projetos comerciais. A solução apresentada não implica em custo aos clientes finais e oferece um ambiente amigável e flexível para a alteração de textos: a utilização de componentes especializados em editoração, no caso, o componente WPTools. Um editor básico é desenvolvido e esse posteriormente é utilizado como um gerador de modelos que são utilizados posteriormente na geração de mala direta. O editor permite a inclusão de imagens, formatação de texto, inclusão de tabelas e até mesmo correção ortográfica.

Um TMainMenu é adicionado, nele criamos uma menu Arquivo e acrescentamos dois subitens a ele. Depois adicionamos um controle WPToolBar, localizado na guia WPTools, e mudamos sua propriedade Align para alTop, dessa forma ele fica posicionado no topo, abaixo do menu. O controle WPToolBar é uma barra de ferramentas que exibe atalhos para as funcionalidades de edição de texto. Na sequência adicionamos as réguas das margens (horizontal e vertical), que também são utilizadas para exibir a tabulação. São respectivamente os controles TWPRuler e TWPVertRuler. Para o controle TWPRuler, a propriedade Align também foi modificada para alTop.

O controle que realiza e trata toda edição de texto é o TWPRichText. Dessa forma o adicionamos no formulário e modificamos sua propriedade Align para alClient fazendo com que o mesmo preencha toda a área disponível.

A WPTools também disponibiliza uma coleção de TActions que podem ser utilizadas e reutilizadas para realização de operações comuns, permitindo que o usuário possa, por exemplo, utilizar o comando Copiar através da barra de ferramentas ou através de um menu. Para usufruir dessas ações é preciso adicionar um controle TActionList, da guia Standard, e efetuar um clique duplo sobre o mesmo. Nesse momento o editor desse TActionList se abre para inserção de ações. Ao clicar com o botão direito do mouse sobre ele, dentro da área Actions, podemos escolher a opção New Stardand Action que abre uma janela para escolha de quais ações inserir. Ao percorrer a treeview exibida é possível identificar facilmente algumas que possuem o prefixo "WPT" (**Figura 1**). Essas pertencem à WPTools.

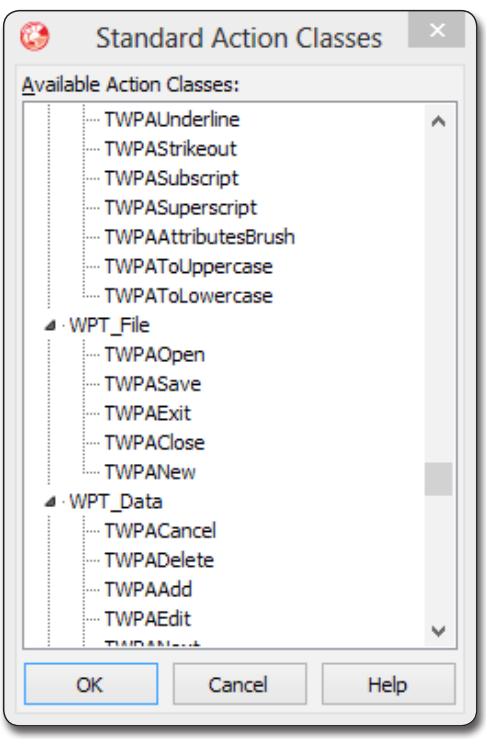

Figura 1. Lista de ações da WPTools

Para que o menu que foi adicionado faça algo relativo à edição de texto, adicionamos as seguintes ações ao TActionList: TWPAOpen, TWPASave, todas sob o nó WPT_File. Então cada ação é ligada a um subitem do menu Arquivo.

Agora que os controles estão em tela, é uma questão de relacioná-los um com outro. Para que as réguas reflitam corretamente o tamanho da página que está em edição é preciso ajustar no controle WPRichText a propriedades WPRuler e VRuler para os respectivos controles de régua. Já para a barra de ferramentas, o WPRichText possui a propriedade WPToolBar, onde é necessário apontar o controle existente.

Com todas as ligações realizadas, basta executar o projeto para termos um editor básico em funcionamento (**Figura 2**).

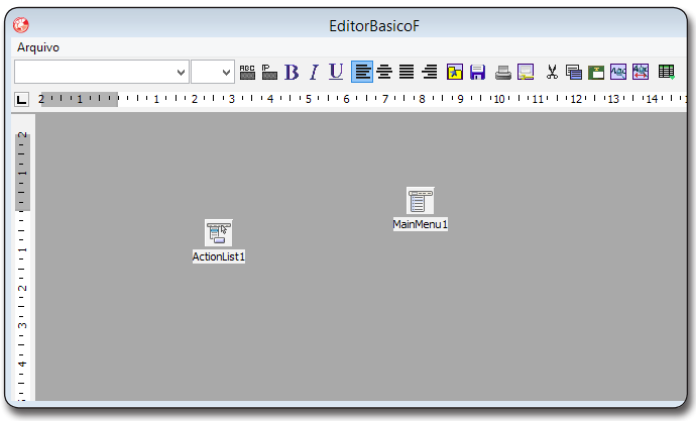

Figura 2. Formulário principal finalizado

Inserindo uma imagem

A barra de ferramenta por padrão vem apenas com alguns botões disponíveis, mas é possível adicionar outros. Ao clicar no controle WPToolBar podemos visualizar via Object Inspector as coleções de botões, como mostra a **Figura 3**. Ao expandir cada categoria é possível ativar ou desativar o que é exibido.

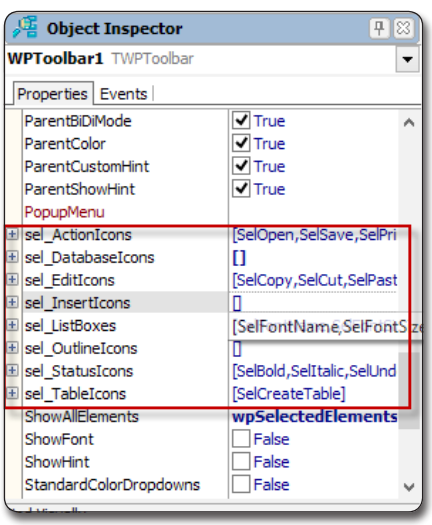

Figura 3. Botões que podem ser exibidos

Vamos incluir então o botão que permite a inserção de uma imagem no texto. Ele está localizado dentro da coleção sel_InsertIcons, com o nome de SelInsImage. Ao executar novamente o editor, será possível incluir uma imagem no texto. Também possível ajustar a forma com que essa imagem se posiciona em relação ao texto. Temos quatro possibilidades: ancorar ao caractere, móvel com quebra automática, móvel com quebra à esquerda e móvel com quebra à direita. Por padrão uma imagem é inserida e ancorada ao texto (caractere).

Movimentando uma imagem

É preciso oferecer meios ao usuário para que ele escolha como a imagem inserida deve se movimentar. Isso é feito através de um menu suspenso que irá aparecer quando o usuário clicar com o botão direito do mouse sobre uma imagem.

Esse menu é um TPopupMenu, que possui seus itens como mostrado na **Figura 4**.

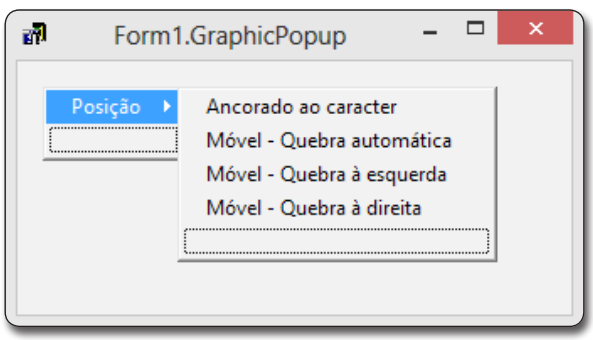

Figura 4. Popup de posicionamento de imagem

Para responder ao evento do clique com o botão direito, o componente TWPRichText disponibiliza o evento OnTextObjectMouseDown. Esse é disparado quando um elemento da WPTools recebe um clique de mouse.

Quando uma imagem é inserida no texto, a visualização apresentada nada mais é do que um objeto interno da WPTools, ou seja, a imagem transforma-se nesse objeto. Dessa forma é possível manter uma referência a ele e reutilizar essa referência para mudar, no caso de uma imagem, seu posicionamento.

Para isso é adicionada uma variável na seção private do formulário, chamada CurrentPTextObj, do tipo PTTextObj. Esse é o tipo que representa um objeto inserido em um parágrafo. Para iniciar seu uso, implementamos então o evento OnTextObjectMouseDown citado anteriormente conforme mostra a **Listagem 1**.

O evento retorna em seus parâmetros uma variável pobj, que representa o objeto que foi clicado no parágrafo. Na linha 5 foi definida uma variável do tipo de TPoint para armazenar a posição do clique do mouse. Talvez a existência dos parâmetros X e Y possam causar dúvida quanto à necessidade dessa variável, pois pode-se imaginar que X e Y retornariam a posição no clique, mas não, X e Y retornam na verdade a posição do objeto que sofreu o clique. Na linha 7 é então armazenada uma referência a esse objeto e, caso tenha sido clicado com o botão direito, como é testado na linha 8, a posição do clique é retornada pela função GetCursorPos, que na verdade retorna a posição atual do cursor do mouse. Então, de posse dessa posição, o TPopupMenu é exibido (linha 11).

Cada item desse TPopupMenu executa um mesmo método da WPTools responsável configuração de posicionamento, como mostra a **Listagem 2**.

Listagem 1. Evento OnTextObjectMouseDown

```
01 procedure TForm1.WPRichText1TextObjectMouseDown(Sender: TWPCustomR-
tfEdit;
02 pobj: TWPTextObj; obj: TWPObject; Button: TMouseButton;
  Shift: TShiftState; X,
03 Y: Integer);
04 var
05  p: TPoint;
06 begin
07  CurrentPTextObj := pobj;
08  if ssRight in          Shift then
09  begin
10    GetCursorPos(p);
11    GraphicPopup.Popup(p.x,p.y);
12  end;
13 end;
```

Listagem 2. Eventos do TPopupMenu

```
01 procedure TForm1.asCharacter1Click(Sender: TObject);
02 begin
03  if CurrentPTextObj<> nil then
04  begin
05   WPRichText1.TextObjects.ChangePositionMode(CurrentPTextObj, wpotPar);
06   WPRichText1.Invalidate;
07  end;
08 end;

09 procedure TForm1.Auto1Click(Sender: TObject);
10 begin
11  if CurrentPTextObj<> nil then
12  begin
13   WPRichText1.TextObjects.ChangePositionMode
    (CurrentPTextObj, wpotPar, wpwrAutomatic);
14   WPRichText1.Invalidate;
15  end;
16 end;

17 procedure TForm1.MovableGraphicWrapLeft1Click(Sender: TObject);
18 begin
19  if CurrentPTextObj<> nil then
20  begin
21   WPRichText1.TextObjects.ChangePositionMode
    (CurrentPTextObj, wpotPar, wpwrLeft);
22   WPRichText1.Invalidate;
23  end;
24 end;

25 procedure TForm1.MovableGraphicWrapRigth1Click(Sender: TObject);
26 begin
27  if CurrentPTextObj<> nil then
28  begin
29   WPRichText1.TextObjects.ChangePositionMode(CurrentPTextObj,
   wpotPar, wpwrRight);
30   WPRichText1.Invalidate;
31  end;
32 end;
```

Como pode ser visto nas linhas 5, 13, 21 e 29, o método executado é o mesmo, o ChangePositionMode, o que muda são seus parâmetros. O valor wpotBar utilizado indica que a imagem deve ficar ancorada ao parágrafo e os valores wpwrAutomatica, wpwrLeft e wpwrRight configuram a movimentação dessa imagem. Já a chamada do método WPRichText1. Invalidade é necessária para forçar a atualização de visualização do componente. Execute a aplicação nesse momento, insira uma imagem em um texto e veja seu comportamento com cada configuração.

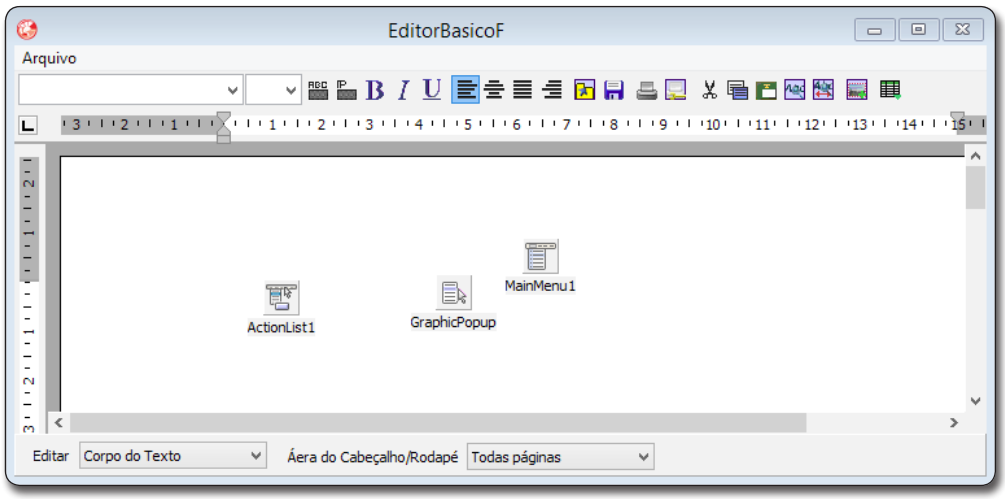

Figura 5. Editor Básico

Cabeçalho e rodapé

A WPTools também disponibiliza a criação de cabeçalhos e rodapés, para facilitar o acesso a esse recurso vamos colocar um painel posicionado em baixo da área de edição e nele adicionar dois TComboBox, como mostra a **Figura 5**.

O primeiro TComboBox (comboEditar) é utilizado para definir qual parte do texto será editada. Ele contém os itens: Corpo do Texto, Cabeçalho e Rodapé. Já o segundo TComboBox (comboArea) é utilizado para determinar quando esse cabeçalho ou rodapé devem aparecer. Ele contém os itens: Todas as páginas, Páginas ímpares, Páginas pares e Primeira página.

Para mover o foco de edição para o cabeçalho é preciso codificar o evento OnChange do controle comboArea, como mostra a **Listagem 3**.

Ao selecionar um item do comboEditar, o índice selecionado é transformado em um valor do tipo TWPPagePropertyKind (linha 3). Esse tipo contém o valores wpIsBody, wpIsHeader e wpIsFooter, nessa exata ordem. Assim, o elemento 0 (zero) do comboEditar corresponde ao elemento 0 (zero) do tipo necessário e conseguimos acessar dinamicamente essas partes do texto. Para que o tipo TWPPagePropertyKind seja reconhecido é necessário adicionar a unit WPRTEDefsConsts na seção uses.

Para ajustar quando o cabeçalho deve aparecer é preciso codificar o evento OnChange do ComboArea, como mostra a **Listagem 4**.

Na linha 2 transformamos o índice selecionado controle ComboArea em um valor do tipo TWPPagePropertyRange, que representa onde o cabeçalho/rodapé irá aparecer. Uma vez que um cabeçalho ou rodapé estão inseridos no texto, é possível navegar entre eles sem utilizar o comboEditar, bastando clicar duplamente sobre o cabeçalho ou rodapé. Então, caso isso ocorra é preciso fazer com que o comboEditar reflita exatamente qual a área do texto que está em edição. Isso pode ser feito pelo evento OnWorkOnTextChanged do componente TWPRichText, como mostra a **Listagem 5**.

Na linha 3 o controle comboEditar é atualizado para mostrar a área em edição. Transformamos o valor contido na propriedade WorkOnText para Integer, representando um índice. O mesmo na linha 4, agora indicando como o cabeçalho/rodapé é exibido. Contudo, se não estamos editando um cabeçalho ou rodapé, não há necessidade de escolher a área, então a partir da linha 5 isso é tratado.

Listagem 3. Selecionando o cabeçalho

```
01 procedure TForm1.comboEditarChange(Sender: TObject);
02 begin
03   WPRichText1.WorkOnText := TWPPagePropertyKind(ComboEditar.ItemIndex);
04   WPRichText1.SetFocus;
05 end;
```

Listagem 4. Visualização do Cabeçalho/Rodapé

```
01 procedure TForm1.ComboAreaChange(Sender: TObject);
02 begin
03  WPRichText1.HeaderFooterTextRange :=
    TWPPagePropertyRange(ComboArea.ItemIndex);
04  WPRichText1.SetFocus;
05 end;
```

Listagem 5. Refletindo a área de edição

```
01 procedure TForm1.WPRichText1WorkOnTextChanged(Sender: TObject);
02 begin
03  comboEditar.ItemIndex := Integer(WPRichText1.WorkOnText);
04  comboArea.ItemIndex := Integer(WPRichText1.HeaderFooterTextRange);
05  ComboArea.Enabled := WPRichText1.WorkOnText<>wpisBody;
06  if WPRichText1.WorkOnText<>wpisBody then
07    ComboArea.Color := clWindow
08  else ComboArea.Color := clBtnFace;
09 end;
```

Visualização do texto

O texto que é visualizado dentro do controle WPRichEdit também pode receber um zoom de visualização. Adicionamos um TSpinEdit (editZoom) ao painel inferior, com sua propriedade Value configurada para 100. Então em seu evento OnChange mudamos alteramos a visualização como mostra a **Listagem 6**.

Na linha 3 o valor contido no TSpinEdit é passado à propriedade Zooming do TWPRichText, influenciando a exibição do texto.

Além de zoom a visualização também possui layouts - Normal e Impressão. Para permitir que se alterne entre esses tipos, adicionamos um TPopupMenu com os itens Normal e Impressão inseridos. Depois, respectivamente no evento OnClick de cada item inserimos o código da **Listagem 7**.

Listagem 6. Modificando o Zoom

```
01 procedure TForm1.EditZoomChange(Sender: TObject);
02 begin
03   WPRichText1.Zooming := EditZoom.Value;
04 end;
```

Listagem 7. Modificando o Layout

```
01 procedure TForm1.Normal1Click(Sender: TObject);
02 begin
03   WPRichText1.LayoutMode := wplayNormal;
04 end;
05 procedure TForm1.Impresso1Click(Sender: TObject);
06 begin
07   WPRichText1.LayoutMode := wplayFullLayout;
08 end;
```

As linhas mais importantes são a 3 e 7. Nelas alteramos o valor do modo layout do componente WPRichText. Agora adicionamos um TButton ao TPanel inferior, modificamos sua propriedade Caption para Layouts, a propriedade Style alteramos para bsSplitButton e em DropDownMenu apontamos para o TPopupMenu recém inserido. Assim conseguimos criar um botão suspenso com as opções de layout. Execute novamente a aplicação e veja a mudança de layout funcionando.

Correção ortográfica

Todo editor de texto que se preze precisa de um corretor ortográfico. O WPTools possui um componente que pode ser adquirido a parte para realizar essa função, contudo, o mesmo não possui um dicionário em português do Brasil. A solução é integrar um dicionário externo ao editor.

A empresa ec-software criou uma versão para Delphi do dicionário NHunspell (ver seção **Links**). Esse dicionário é largamente utilizado pela comunidade open-source, sendo o corretor ortográfico oficial de projetos como Open Office e Firefox.

O corretor consiste de uma DLL, Hunspellx86.dll, e as units NHunspell, NHunXml e PasZip. Para simplificar o processo de integração com a WPTools, eu mesmo criei uma unit que adapta as funções existentes para as funções exigidas da WPTools. O pacote completo está em um arquivo zip chamado WPNHunspell disponível para download com este artigo.

Uma vez que realizou o download desse arquivo, o descompacte para uma pasta qualquer, por exemplo, em C:\WPNHunspell. Existirão duas pastas, Source e Demo, que respectivamente são o código fonte da integração e um projeto de mostra como utilizar o corretor.

Para utilizar o corretor em nosso editor básico, adicionamos no Search path do projeto (Project > Options) a pasta source do WPNHunspell. Feito isso o projeto consegue agora ter acesso às Units adaptadas.

Agora podemos adicionar na seção uses a única unit requerida para integração, a unit WPNHunspellU. Mas como se dará a correção do texto? A integração irá permitir que ao digitar as palavras, as mesmas sejam verificadas. Isso é chamado de Spell as Go. Também será possível clicar com o botão direito do mouse e escolher em um menu a correção para aquela palavra inválida. Também será possível executar uma caixa de diálogo que percorrerá todo o texto em busca das palavras desconhecidas.

Para começar adicionamos um TPopupMenu (popCorrecao), nele é preciso adicionar um item de menu. Por exemplo, esse item pode ter o Caption como Correção Ortográfica e sua propriedade Name para itemCorrecao. Ligamos esse componente popCorrecao ao editor, através da propriedade WPRichText1.PopupMenu. Então no evento OnShow do formulário principal adicionamos a seguinte linha de código:

```
ConfigureWPNHunspell(WPRichText1, itemCorrecao);
```

Esse procedimento realiza toda amarração necessária entre seu componente TWPRichText e o item de menu que irá exibir a sugestão de palavras. Depois, no evento OnClose do formulário principal executamos uma chamada ao procedimento UnconfigureWPNHunspell. Esse desfaz toda a amarração.

Para acionar o diálogo de verificação ortográfica é necessário apenas adicionar um botão no painel inferior e no seu evento OnClick chamar o procedimento DoSpellCheck. Para encerrar a integração, basta disponibilizar na mesma pasta do executável dois arquivos necessários, o dicionário em português do Brasil (pt_br.oxt) e a DLL do Hunspell (Hunspellx86.dll), ambos disponíveis no arquivo compactado WPNHunspell anteriormente citado. Basta executar a aplicação e ver a correção ortográfica acontecendo com o mínimo de código. A **Figura 6** mostra a janela de correção em ação.

Mesclagem de informações

A necessidade real de um editor de textos em um aplicativo está muitas vezes ligada à geração de mala direta, de contratos. A WPTools oferece suporte a esse tipo de operação. Mesclagem significa a atualização automática de campos em qualquer documento. Isso é implementado através do procedimento MergeText e do evento OnMailMergeGetText. Esses campos podem ser adicionados através do método InputMergeField.

Basicamente ao executar o procedimento MergeText o componente TWPRichText localiza todos os campos contidos no texto e dispara o evento OnMailMergeGetText para cada um deles. Esse evento então é usado para preencher os dados do campo ou ler seu conteúdo. Esses campos não são destruídos após o processo de mesclagem, permitindo trocar quantas vezes forem necessárias seu conteúdo. Podem ser mesclados textos, textos com formatação e

imagens. Por texto com formatação inclui-se HTML, RTF ou o próprio formato WPTools.

Para demonstrar seu uso vamos criar uma situação de emissão de cartas onde os dados de uma pessoa (nome, endereço, bairro, cidade, UF e CEP) devem ser mesclados ao conteúdo dessa carta e então essa pode ser impressa.

No projeto do editor adicionamos no TMainMenu um item chamado Campos, esse chamará um novo formulário (CamposF) a ser criado. Esse formulário tem sua propriedade FormStyle modificada para fsStayOnTop, isso para que ele permaneça sempre sobre qualquer outro formulário. Nele também é adicionado um controle TListBox (listCampos) com sua propriedade Align configurada para alClient, para preencher o formulário. Nesse TListBox são inseridos os campos de mesclagem disponíveis, como mostra a **Figura 7**.

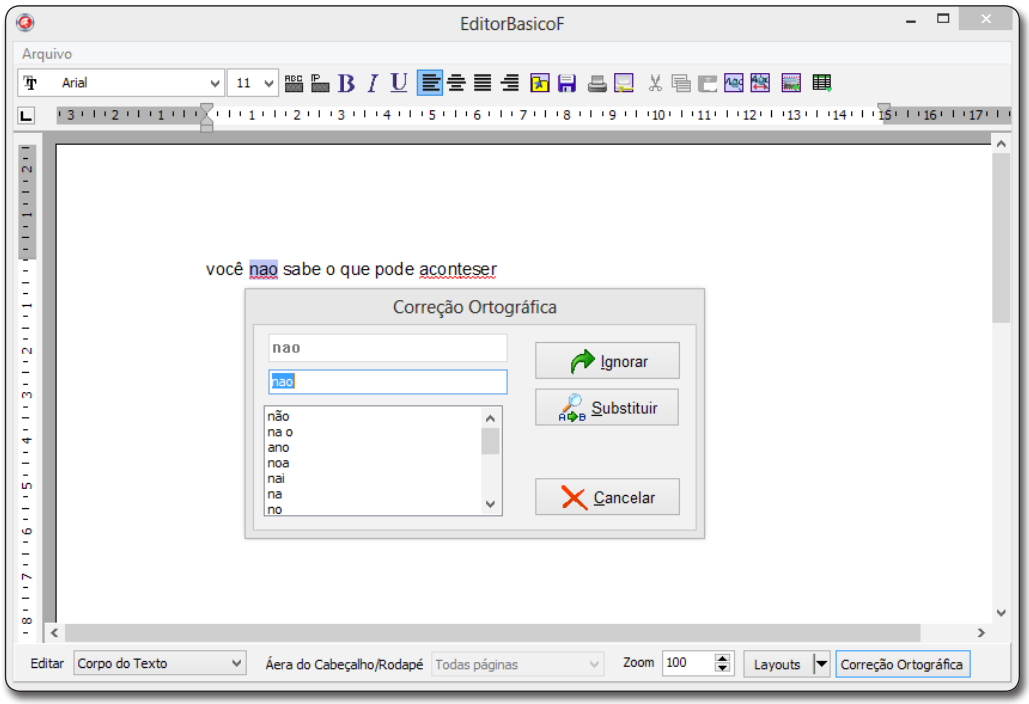

Figura 6. Correção Ortográfica em execução

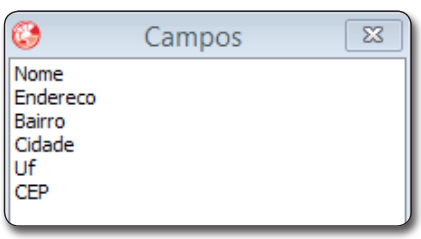

Figura 7. Campos de mesclagem

Como por padrão no Delphi um formulário criado é colocado automaticamente na lista de auto criação (Project > Options > Forms), vamos modificar o método Show da classe TForm, acrescentando um parâmetro que permita ao formulário de campos inserir no editor o campo escolhido. Dessa forma no formulário de campos criamos uma procedure na seção public e a implementamos como mostra a **Listagem 8**.

Listagem 8. Novo método Show

```
01 public
02   procedure Show(WPTEditor: TWPRichText); reintroduce;
03 // implementação
04 procedure TCamposF.Show(WPTEditor: TWPRichText);
05 begin
06   FEditor := WPTEditor;
07   inherited Show;
08 end;
```

Na linha 2 o comando reintroduce indica ao compilador que o método Show original da classe TCamposF deve ser substituído por esse recém criado. Isso é refletido já no próprio IDE, que deixa de sugerir o método Show original, herdado da classe TForm. O parâmetro WPTEditor é armazenado em um campo privado (linha 6) que precisa ser criado também. Dessa forma é possível então implementar o evento OnDblClick do TListBox como é mostrado na **Listagem 9**.

Na linha 5 verificamos se o campo FEditor possui uma instância associada a ele. Se possuir então definimos que o valor da variável nomeCampo é o item que foi selecionado no TListBox.

Listagem 9. Inserindo um campo

```
01 procedure TCamposF.listCamposDblClick(Sender: TObject);
02 var
03  nomeCampo: string;
04 begin
05   if FEditor <> nil then
06   begin
07     nomeCampo := listCampos.Items[listCampos.ItemIndex];
08     FEditor.InputMergeField(nomeCampo,nomeCampo);
09   end;
10 end;
```

Esse valor então é utilizado no método InputMergeField, que recebe dois parâmetros: o nome interno do campo e o texto que aparecerá visualmente quando esse é inserido. Isso deixa o formulário CamposF pronto para inserção dos campos definidos. Para chamá-lo, no evento OnClick do item de menu Campos, inserido anteriormente, executamos o código da **Listagem 10**.

Na linha 3 temos o novo método Show em execução, recebendo a instância do editor em uso. Execute o editor e insira os campos. A **Figura 8** mostra o resultado dessa operação.

É possível identificar que automaticamente os campos são destacados por chaves e, além disso, eles podem receber formatação como qualquer outro texto. Salve esse documento em uma pasta de fácil acesso, com o nome de modelo.rtf.

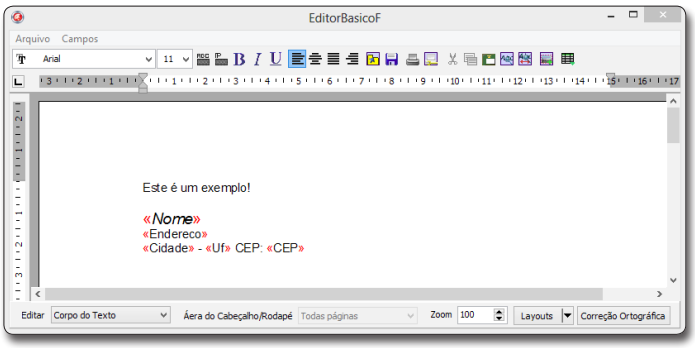

Figura 8. Texto com campos inseridos

Listagem 10. Chamando o formulário de campos

```
01 procedure TForm1.Campos1Click(Sender: TObject);
02 begin
03   CamposF.Show(WPRichText1);
04 end;
```

O editor criado até o momento funciona como um editor de modelos. Ele pode ser utilizado para criar os mais diversos modelos um sistema possa necessitar, desde que ele seja constantemente atualizado com a lista de campos disponíveis.

Para realizar a mesclagem vamos criar um novo aplicativo, que representa um aplicativo comum que detém o conteúdo que precisa ser mesclado. Então criamos um novo projeto do tipo VCL Forms (File > New > VCL Forms Application). Nesse formulário incluímos um TClientDataSet (cdsClientes) e um TDaSource (dsClientes) ligado a ele. Também inserimos um TDBGrid (gridClientes) e o ligamos ao TDataSource existente. No componente cdsClientes inserimos os mesmos campos existentes para mesclagem, de tal forma que o usuário posso ele mesmo cadastrar os clientes. Incluímos também um botão Imprimir que irá realizar a mesclagem e imprimir os documentos. A **Figura 9** mostra esse formulário em execução.

Quem realiza a mesclagem é o controle TWPRichText, então incluímos um nesse formulário e modificamos sua propriedade Visible para false. Então implementamos seu evento OnMailMergeGetText como mostra a **Listagem 11**.

O parâmetro inspName retorna o nome do campo que sofrerá a mesclagem, enquanto que o Contents permite passar o valor para esse campo.

CURSOS ONLINE

A Revista Java Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

CONHEÇA ALGUNS DOS CURSOS:

- Curso de noSQL (Redis) com Java
- Curso Básico de JDBC
- Java Básico: Aplicações Desktop
- JSF com Primefaces
- Conhecendo o Apache Struts

Para mais informações :
www.devmedia.com.br/curso/javamagazine
(21) 3382-5038

DEVMEDIA

Listagem 11. Mesclagem

```
01 //incluir no Uses: WPRTEEdit
02 procedure TCadastroF.WPRichText1MailMergeGetText(Sender: TObject;
03   const inspname: string; Contents: TWPMMInsertTextContents);
04 begin
05  if inspname = 'Nome' then
06  begin
07   Contents.StringValue := cdsClientesNome.AsString;
08   Exit;
09  end;
10  if inspname = 'Endereco' then
11  begin
12   Contents.StringValue := cdsClientesEndereco.AsString;
13   Exit;
14  end;
15  if inspname = 'Bairro' then
16  begin
17   Contents.StringValue := cdsClientesBairro.AsString;
18   Exit;
19  end;
20  if inspname = 'Cidade' then
21  begin
22   Contents.StringValue := cdsClientesCidade.AsString;
23   Exit;
24  end;
25  if inspname = 'Uf' then
26  begin
27   Contents.StringValue := cdsClientesUf.AsString;
28   Exit;
29  end;
30  if inspname = 'CEP' then
31  begin
32   Contents.StringValue := cdsClientesCEP.AsString;
33   Exit;
34  end;
35 end;
```

Listagem 12. Disparando a mesclagem

```
01 procedure TCadastroF.Button1Click(Sender: TObject);
02 begin
03  cdsClientes.First;
04  while not cdsClientes.Eof do
05  begin
06   WPRichText1.Clear;
07   WPRichText1.LoadFromFile('d:\modelo.rtf');
08   WPRichText1.InsertPointAttr.Hidden := true;
09   WPRichText1.MergeText;
10   WPRichText1.Print;
11   cdsClientes.Next;
12  end;
13 end;
```

Figura 9. Cadastro de clientes para mesclagem

Todo processo é uma série de decisões para verificar qual é o campo do momento.

Para colocar isso em execução, no botão Imprimir carregamos o modelo e percorremos o cdsClientes disparando o método MergeText. A **Listagem 12** mostra esse código.

A cada cliente percorrido o documento é carregado (linha 7), e com a execução do método MergeText (linha 9) os dados são transportados para o documento. Já a linha 10 imprime o documento depois de mesclado. O comando da linha 8 informa à WPTools para ocultar os caracteres de campo de mesclagem, sem isso, as marcações "<<" e ">>" também seriam impressas junto de cada campo.

Com este artigo abordamos um aspecto pontual do uso de mala direta e edição de textos em um aplicativo Delphi. A solução pode ser aprimorada com um cadastro de modelos, ou até mesmo com a visualização da mesclagem em tela, ao invés de ir para a impressora. O importante é que existe uma solução pronta e de alta qualidade para seus clientes. Basta aprender a usufruir de seus recursos.

Autor

Paulo Quicoli
pauloquicoli@gmail.com
Editor da revista ClubeDelphi. Formado em processamento de dados pela FATEC-TQ. Atua como analista programador de sistemas da Siplan Control-M (www.siplancontrolm.com.br) unidade Jaboticabal, e professor da FATEC, campus Taquaritinga.
Blog: http://quicoli.worpress.com

Links:

Download da versão Trial da WPTools
http://wptools.com/products/wptools/index.htm

NHUnspell
http://www.ec-software.com/downloads_delphi.html

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Delphi e WCF: aplicações multicamadas

Criando aplicações clientes em Delphi para servidores C# .NET com WCF Data Services e Entity Framework

Os SaaS estão se tornando cada vez mais comuns dentro das empresas devido aos grandes benefícios que sua utilização pode trazer, tais como diminuição de custos e acesso fora do ambiente empresarial. Porém, para que eles atinjam seus objetivos, necessitam de uma série de requisitos para atender seus usuários, visto que geralmente, são centenas ou até milhares de conexões simultâneas solicitando recursos de um mesmo servidor. Por isso, uma arquitetura de SaaS deve estar pronta para atender a esse tipo de demanda, e a escolha de determinadas tecnologias para seu desenvolvimento é de suma importância para atingir esses objetivos.

Um tipo de arquitetura que se torna natural ao desenvolvimento de SaaS é o modelo multicamadas, pois ele apresenta praticamente todas as características necessárias para a construção dos mesmos. Esse modelo é caracterizado basicamente pelo fato de que a aplicação cliente jamais acessa diretamente o banco de dados como no modelo cliente servidor tradicional (duas camadas). Em suma, pelo menos três camadas são necessárias, ou seja: (I) banco de dados, (II) servidor de aplicação e (III) apresentação. A camada do banco de dados, como o próprio nome sugere, é onde ficará armazenado o SGBD, enquanto que na segunda camada ficarão todas as regras de negócios e a lógica de acesso aos dados. Por fim, a camada de apresentação é a que fornecerá a interface para que o usuário interaja com o sistema e realize as operações. Fazendo uma analogia deste modelo com a aplicação exemplo desenvolvida neste artigo, nossa primeira camada será o SQL Server, a segunda será um servidor .NET WCF com REST e finalmente, a última camada será constituída de uma aplicação cliente em Delphi que fará acesso ao servidor WCF.

Fique por Dentro

Este artigo apresenta o desenvolvimento de um sistema multicamada que faz uso de conceitos como SaaS (Software como Serviço) e que pode interoperar com qualquer linguagem de programação. Desta forma, será utilizado como backend e regras de negócio um servidor WCF (Windows Communication Foundation) Data Services do .NET Framework utilizando os conceitos de REST (Representational State Transfer) e como frontend uma aplicação em Delphi VCL. O servidor irá acessar uma base de dados SQL Server, expondo através de métodos os dados ali existentes. Todo o acesso a esse banco de dados é realizado através do Entity Framework, que é a ferramenta gratuita de mapeamento relacional da Microsoft. A aplicação cliente irá consumir os serviços expostos por esse servidor, realizando um mapeamento entre esses dados e um TClientDataSet e através do recurso de Live Bindings irá carrega-los em um formulário para permitir sua edição.

A arquitetura multicamadas apresenta uma série de vantagens sobre o modelo cliente servidor tradicional, como por exemplo: modularização, facilidade de redistribuição, clientes leves, economia de conexões ao banco de dados, escalabilidade, independência de localização, linguagem de programação e SGDB. Podemos notar que este modelo apresenta muitos benefícios, podendo-os utilizar a nosso favor no desenvolvimento de arquiteturas para SaaS. De todas essas características, a que mais está relacionada ao escopo deste artigo é a independência de linguagem de programação, na qual será dado maior foco neste momento.

Para que exista independência entre linguagens de programação, é necessário existir uma camada de software entre a aplicação e o sistema em questão chamada middleware, o qual tem o objetivo básico de converter os dados de um formato para outro com o intuito de garantir a compatibilidade entre ambos. Existem diversas tecnologias de middleware no mercado atualmente. O próprio DataSnap do Delphi atualmente pode entrar nesta categoria, visto

que nas últimas versões do IDE essa tecnologia ganhou independência de protocolos proprietários, fato que possibilita a criação de frontends nas mais diversas linguagens de programação.

Além disso, outro modelo que vem sendo bastante utilizado é a disponibilização de serviços utilizando os conceitos de REST, como uma alternativa aos já conhecidos Web Services com o protocolo SOAP. Esse padrão especifica que os recursos do servidor são acessados por um endereço único utilizando somente URLs, o que facilita o que é chamado de "descoberta" de recursos do servidor. Essa técnica será bastante discutida e demonstrada na aplicação prática desenvolvida neste artigo, na qual por meio do Delphi estaremos acessando um servidor WCF com REST.

Como citado anteriormente, o .NET Framework é considerado um middleware e a tecnologia por ele utilizada para a exposição de serviços e recursos é o WCF Data Services, o qual oferece suporte aos padrões SOAP e REST.

Criando o banco de dados

O primeiro passo para a construção de nosso exemplo é a criação do banco de dados a ser utilizado na aplicação, o qual recebeu o nome de ESTOQUE e será criado no SGBD SQL Server. Tal banco é constituído por duas tabelas, uma para armazenar as categorias e outra para armazenar os produtos. A **Listagem 1** apresenta a instrução SQL utilizado para a definição da tabela CATEGORIAS.

Observe que a tabela possui somente dois campos, o nome da categoria e uma chave primária auto incremento. **A Listagem 2** ilustra o SQL para a criação da tabela PRODUTOS.

Note que a tabela possui uma chave estrangeira relacionada com a tabela de Categorias. Essa ligação será posteriormente utilizada para o desenvolvimento do relacionamento mestre-detalhe.

Criando o servidor WCF no Visual Studio

O segundo passo para definição de nossa arquitetura é a camada do servidor de aplicação, o qual conterá todas as regras de acesso a dados, as classes mapeadas do banco de dados e também as regras de negócio. Para isso, um novo projeto deve ser iniciado no Visual Studio (File>New>Project). Na tela denominada New Project, deve-se ter atenção para que seja escolhida uma aplicação do tipo Visual C# e o objeto a ser criado deve ser uma Class Library.

Este tipo de projeto irá gerar um assembly no formato de DLL, no qual ficará embutido todo o modelo de acesso aos dados. É necessária essa definição da Class Library, visto que o framework .NET permite que as operações da DLL sejam disponibilizadas e/ou consumidas seja por um Web Service, serviço, aplicação MVC ou até mesmo localmente em uma aplicação desktop.

Criando um Data Model no ADO.NET Entity Framework

Para fazer a persistência objeto relacional de nossa aplicação será utilizado o Entity Framework da tecnologia .NET, o qual tem o objetivo de fazer o mapeamento do banco de dados relacional na forma de objetos. Com isso, o desenvolvedor não necessita se preocupar com códigos em linguagem SQL, tais como comandos CRUD (select, insert, update e delete).

Listagem 1. SQL para a criação da tabela Categorias

```
CREATE TABLE CATEGORIAS (
                IDCATEGORIA INTEGER NOT NULL IDENTITY,
                NOME VARCHAR(50) NOT NULL,
                CONSTRAINT PK_CATEGORIAS PRIMARY KEY (IDCATEGORIA)
);
```

Listagem 2. SQL para a criação da tabela Produtos

```
CREATE TABLE PRODUTOS (
                IDPRODUTO INTEGER NOT NULL IDENTITY,
                NOME VARCHAR(200) NOT NULL,
                IDCATEGORIA INTEGER,
                CONSTRAINT PK_PRODUTOS PRIMARY KEY (IDPRODUTO),
                CONSTRAINT FK_PRODUTOS_CAT FOREIGN KEY (IDCATEGORIA)
REFERENCES CATEGORIA (IDCATEGORIA)
);
```

Dentro do IDE do Visual Studio, um novo item deve ser adicionado ao projeto (Project>Add New Item), o qual deve ser do tipo ADO.NET Entity Data Model. Na próxima tela deve ser escolhida a opção "Generate from database", a qual tem o intuito de gerar o modelo de classes persistentes baseado em um banco de dados já existente. É possível também, através do diagrama de classes, fazer a geração automática de todo o banco de dados a partir de um modelo (abordagem Model First).

O próximo passo é configurar a conexão com o banco de dados, e após, selecionar as duas tabelas criadas anteriormente. Depois dessa sequência de passos, o modelo de acesso aos dados já estará pronto, tendo como resultado o que mostra a **Figura 1**. Após isso, é importante compilar a solução através do menu Build>Build Solution para verificar se tudo está certo.

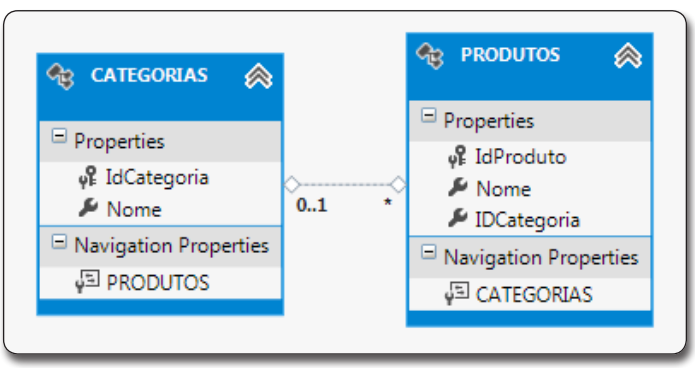

Figura 1. Data Model no Entity Framework

Pode-se notar que as classes CATEGORIAS e PRODUTOS possuem duas características: properties e navigation properties. A primeira simplesmente é a representação dos campos das tabelas em forma de atributos, enquanto que a segunda são as propriedades de navegação, ou seja, a propriedade PRODUTOS da classe CATEGORIAS é uma coleção de todos os produtos pertencentes a esta categoria. Por outro lado, a propriedade CATEGORIAS da classe PRODUTOS apresenta um objeto do tipo CATEGORIA, como se fosse a simulação de um join feito no banco de dados em linguagem SQL. Como comentado anteriormente, por meio dessas

características o Entity Framework abstrai do desenvolvedor todos os comandos SQL.

Outra vantagem desta abordagem é a centralização, ou seja, caso alguma alteração seja feita no modelo de classes orientado a objetos dentro do Visual Studio, ela pode ser replicada automaticamente para o banco de dados. Por outro lado, alterações feitas na base de dados podem também serem replicadas para o modelo orientado a objetos.

Expondo o modelo via WCF Data Services

Como o Data Model está dentro de uma Class Library, é possível disponibiliza-lo como um serviço para que possa ser consumido por múltiplos dispositivos através da Internet e/ou rede local. Iremos então criar um novo Web Site que irá hospedar nossa DLL (botão direito no Solution Explorer>Add>New Web Site>ASP.NET Empty Web Site Visual C#). A **Figura 2** mostra como deverá ficar a tela do Visual Studio com os dois projetos já definidos.

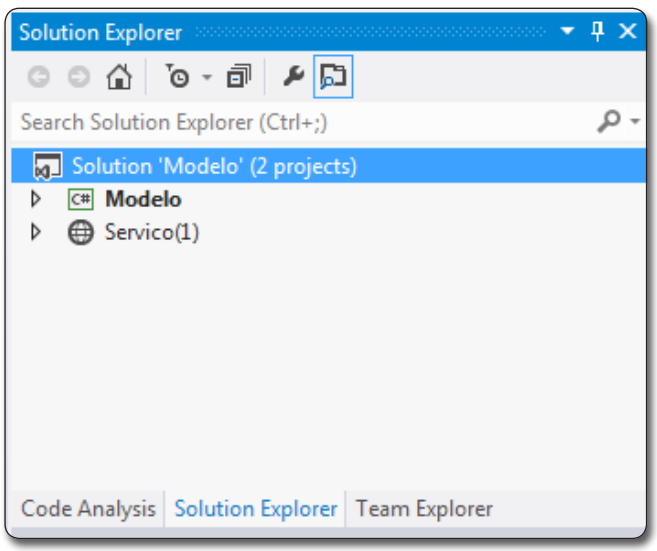

Figura 2. Solution Explorer

Nota

O Visual Studio trabalha com o mesmo conceito de ProjectGroup que existe no Delphi, o que permite adicionar mais de um projeto na mesma "solução", como mostrado na **Figura 2**. Caso o item superior da árvore não esteja aparecendo, acesse Tools>Options>Projects and Solutions>Always show solution. Esta opção deve estar habilitada, caso contrário o procedimento anterior não poderá ser realizado.

O próximo passo é adicionar nosso modelo de dados dentro do Web Site acessando WebSite>Add Reference>Solution>Projects e escolhendo o projeto criado anteriormente. Esse processo é necessário para que tenhamos acesso ao que está definido nas classes mapeadas do banco de dados. Outro procedimento a ser realizado é a adição no website do WCF Data Service (WebSite>Add New Item>Visual C#>WCF Data Service).

O objetivo deste componente é unificar em uma mesma infraestrutura os protocolos de comunicação para objetos distribuídos, ou seja, ao invés de utilizar vários protocolos numa mesma aplicação (tais como Web Services, SOAP, Message Queue, COM+) o WCF fornece em sua infraestrutura todo o acesso e configurações necessárias. Uma grande vantagem desta técnica é que ela utiliza os conceitos de REST, ou seja, os recursos do servidor que deverão ser consumidos pelas aplicações clientes podem ser "descobertos" através da URL, utilizando comunicação básica HTTP com os métodos GET e POST.

A **Listagem 3** mostra o código em C# que configura os acessos às classes de nosso modelo.

Listagem 3. Configuração do servidor

```
01 using System;
02 using System.Data.Services;
03 using System.Data.Services.Common;
04 using System.Collections.Generic;
05 using System.Linq;
06 using System.ServiceModel.Web;
07 using Modelo;

08 public class WcfDataService : DataService<Modelo.ESTOQUEEntities>
09 {
10   public static void InitializeService(DataServiceConfiguration config)
11   {
12     config.SetEntitySetAccessRule("*", EntitySetRights.All);
13     config.SetServiceOperationAccessRule("*",
       ServiceOperationRights.All);
14     config.DataServiceBehavior.MaxProtocolVersion =
       DataServiceProtocolVersion.V3;
15   }
16 }
```

Inicialmente foi feita a importação da referência ao nosso Modelo para que tenhamos acesso às classes CATEGORIAS e PRODUTOS (linha 07). Note também que foi passado o parâmetro "Modelo.ESTOQUEEntities" para a classe, indicando ao Data Service que poderá manipular todas as classes anteriormente definidas (linha 08). Dentro do método InitializeService (linha 10) estão definidas permissões de acesso total à todas as classes (EntitySetRights.All – linha 12) e também há um indicativo ao WCF que habilite todos os métodos (select, insert, update e delete) também para todas as classes (ServiceOperationRightsAll – linha 13).

Outra grande vantagem do WCF Data Service é que a definição desses parâmetros exime os desenvolvedores de programarem os métodos CRUD (create, restore, update e delete) para cada tabela, aumentando significativamente a produtividade.

Finalizando, a última configuração a ser realizada em nosso servidor é a adição da string de conexão ao banco de dados no arquivo Web.config. O código da **Listagem 4** mostra a única alteração que deve ser feita.

Testando o acesso via browser

Após termos feito todas as configurações, chegou a hora de testarmos no browser o resultado de nosso servidor WCF acessando

a opção File>View in Browser (ou clique com o botão direito no arquivo de configuração com a extensão ".svc"). O código da **Listagem 5** mostra o XML resultante deste processo.

Nota

Note que na string de conexão da **Listagem 4** está sendo utilizado o servidor local .\SQLEXPRESS. Caso esteja utilizando outro banco de dados ou a instância de seu servidor tenha outro nome, basta fazer a respectiva alteração no arquivo.

Listagem 4. String de conexão

```
...
<connectionStrings>
  <add name="ESTOQUEEntities"
  connectionString="metadata=res://*/Modelo.csdl|res://*/Modelo.ssdl|res:
  //*/Modelo.msl;provider=System.Data.SqlClient;provider connection
  string=&quot;data source=.\SQLEXPRESS;
  initial catalog=ESTOQUE;
  integrated
  security=True;MultipleActiveResultSets=True;
  App=EntityFramework&quot;"
  providerName="System.Data.EntityClient" />
 </connectionStrings>
</configuration>
```

Listagem 5. XML gerado pelo servidor

```
<service xml:base="http://localhost:4592/WcfDataService.svc/">
 <workspace>
  <atom:title>Default</atom:title>
  <collection href="CATEGORIAS">
   <atom:title>CATEGORIAS</atom:title>
  </collection>
  <collection href="PRODUTOS">
   <atom:title>PRODUTOS</atom:title>
  </collection>
 </workspace>
</service>
```

Observe que existem duas tags XML com a propriedade "href", as quais estão relacionadas com as duas classes que foram geradas anteriormente. Neste momento, fica claro a utilização do padrão REST, em que as informações podem ser obtidas através da URL sem a necessidade de implementação de métodos para buscar os registros.

Outro recurso que pode ser utilizado através deste padrão é simplesmente digitar o nome da classe na URL, como por exemplo: "http://localhost:4592/WcfDataService.svc/CATEGORIAS". Este comando irá trazer em formato XML todos os registros desta tabela.

Como XML é um formato não proprietário e aberto, qualquer aplicação poderá fazer uso dos registros, ficando assim mais claro os conceitos de interoperabilidade entre diferentes plataformas e linguagens de programação.

Criando a aplicação cliente em Delphi com acesso ao Servidor WCF

Estando nosso servidor pronto e configurado para receber conexões de diversos dispositivos, vamos agora criar uma aplicação do tipo VCL no Delphi (File>New>VCL Forms Application), a qual fará acesso ao servidor WCF.

Trabalharemos somente com um formulário que conterá um PageControl com duas guias. A primeira guia é mais básica e servirá para mostrarmos como extrair dados do servidor WCF utilizando os conceitos de REST por meio de HTTP GET. Essa guia será composta somente por um TMemo, um TButton e um TEdit, sendo que o objetivo é digitar no Edit o nome da entidade que está exposta pelo servidor e exibir dentro do memo o XML correspondente aos seus dados. Para que isso seja possível, é necessário adicionar no formulário o componente TIdHTTP da paleta Indy, pois é ele o responsável por fazer o acesso HTTP para o servidor. A **Figura 3** mostra a respectiva tela e a **Listagem 6** apresenta o código que deve ser incluído no evento OnClick do botão.

A **Figura 3** mostra um exemplo de execução, na qual estão sendo buscados os dados relativos à entidade PRODUTOS. Observe que o memo é preenchido com o mesmo XML que havia sido retornado pelo browser no passo anterior.

Listagem 6. Acesso ao servidor WCF com Indy

```
...
01 const
02  URI = 'localhost:3312/Servico/WcfDataService.svc';

03 implementation

04  {$R *.dfm}

05  procedure TFrmMain.btnGetClick(Sender: TObject);
06  var
07   res: string;
08  begin
09   res = IdHTTP.Get(URI + txtEntidade.Text);
10   MM.Lines.Text := res;
11  end;
12 end.
```

Note que no código apresentado primeiramente foi definida uma constante chamada URI (linha 02), a qual armazena o endereço do servidor WCF que foi feito o teste no tópico anterior. É importante notar que não utilizamos nenhum componente de acesso a dados do Delphi, visto que estamos conectando em um servidor REST que envia e recebe os dados por meio da URL. O código para recuperar os dados do servidor é bastante simples, bastando invocar o método Get do componente Indy e passando como parâmetro o endereço do servidor concatenado com o nome da entidade que desejamos obter os dados (linha 09). Esse fato mostra fielmente a utilização de REST, em que é possível "descobrir" recursos através da URL.

Outro teste que pode ser executado é buscar os dados da entidade CATEGORIAS, bastando digitar seu nome na caixa de texto.

Definindo os mapeamentos XML

O arquivo XML retornado pelo servidor WCF no passo anterior ainda necessita de uma transformação para que possa ser lido pelo TClientDataSet do Delphi, visto que esse componente faz

a leitura de arquivos XML no formato datapacket. Para isso, é necessário copiar o XML de produtos que foi retornado dentro do componente TMemo e criar um novo arquivo no diretório da aplicação chamado "PRODUTOS.xml".

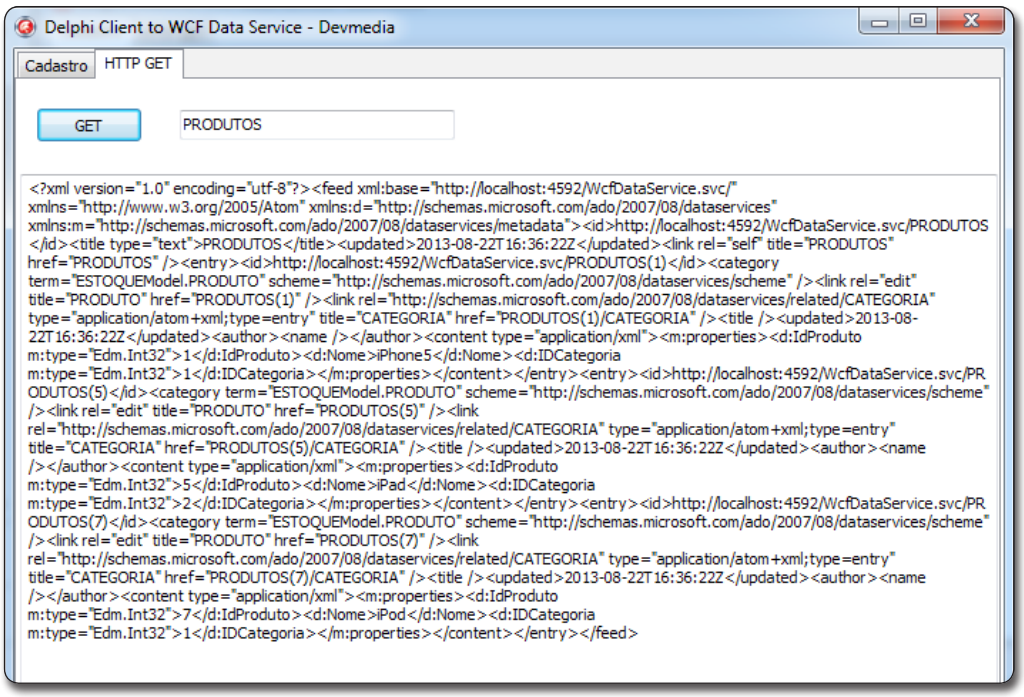

Figura 3. HTTP Get no Delphi

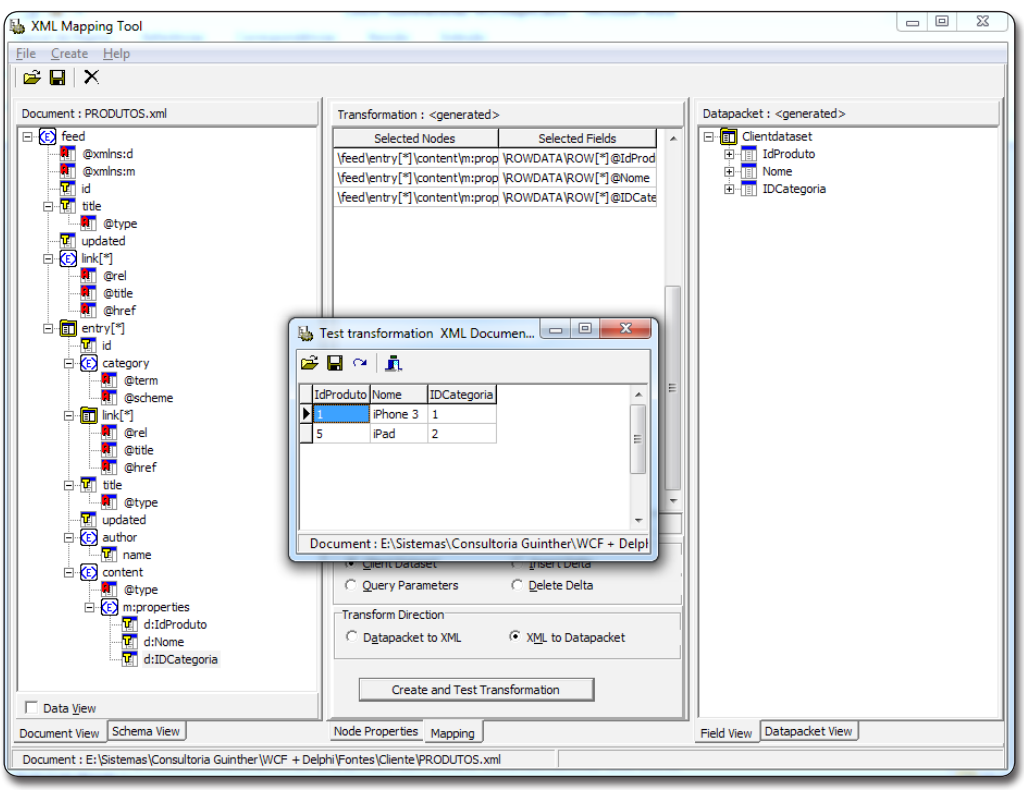

Figura 4. Mapeamento de XML data Datapacket no XML Mapper

Feito isso, este arquivo deve ser aberto no utilitário XML Mapper do Delphi por meio do menu Tools>XML Mapper. Depois, os campos devem ser marcados para sofrerem a transformação clicando duas vezes sobre os nomes "d:IdProduto", "d:Nome" e "d:IDCategoria", levando-os para o painel central da ferramenta, conforme mostra a **Figura 4**.

Na sequência iremos gerar o arquivo PRODUTOS_DP.xml, o qual é um XML já em formato datapacket. Para isso, o botão "Create and Text Transformation" deve ser clicado e depois, quando for mostrada a grade com os dados, clicar em "Save" (a grade é mostrada na **Figura 4**).

O próximo passo é efetivamente criar o mapeamento do XML para datapacket, acessando o menu Create>Datapacket from XML. Observe que no painel direito é mostrado o XML original já transformado em um formato que o TClientDataSet pode acessar (**Figura 4**). Além disso, é necessário criar a transformação por meio do menu Create>Transformation e depois File>Save>Transformation, sendo gerado o arquivo "ToDp.xtr" (certifique-se de que a opção "XML to Datapacket" em "Transform Direction esteja marcada").

Para finalizar é necessário realizar o mapeamento contrário, ou seja, transformar um datapacket que somente o TClienteDataSet entende em um formato XML padrão que o servidor WCF possa consumir. Para isso, a opção "Datapacket to XML" deve estar selecionada e o mesmo procedimento anterior deve ser executado, acessando o menu Create>Transformation e File>Save>Transformation, gerando o arquivo "ToXml.xtr".

Como indicado anteriormente, essa sequência de passos é necessária para que o Delphi consiga interpretar os arquivos XML provenientes do servidor WCF e também o contrário, que o WCF interprete as informações provenientes do Delphi. Os arquivos com a extensão ".xtr" possuem

todas as regras de sintaxe para realizar a conversão entre os dois formatos.

Configurando o acesso ao servidor usando ClientDataSet e XMLTransformProvider

Vamos agora iniciar a construção de nossa aplicação utilizando TClientDataSet e o recurso de Visual Live Bindings do Delphi. O objetivo inicial será construir um formulário básico que no clique do botão, os dados de produtos sejam buscados do servidor WCF e carregados em um TStringGrid. A **Figura 5** mostra a interface visual, sendo os componentes adicionados na outra guia do TPageControl.

Figura 5. Busca dos produtos do servidor WCF

Antes de programarmos essa função, faremos um teste com o XML local para certificar-se de que os dados das transformações com o XML Mapper estão corretos. Será então adicionado no formulário um TClientDataSet, sendo que ele deverá ler o arquivo PRODUTOS_DP.xml através da opção "Load from MyBase Table" clicando com o botão direito no TClientDataSet. Agora faremos o bind visual entre o ClientDataSet e o StringGrid utilizando o recurso de Visual Live Bindings do Delphi, conforme mostra a **Figura 6**. Uma vez feito o bind, os dados do XML são automaticamente mostrados no StringGrid (neste momento também é interessante adicionar os fields no dataset, clicando com o botão direito e escolhendo a opção Add All Fields). É importante salientar que neste momento os dados proveem do arquivo XML local, não tendo ainda conexão com o servidor WCF.

A partir de agora iremos buscar os dados do servidor, utilizando os arquivos "xtr" para a conversão dos dados, como visto anteriormente. O primeiro passo é limpar as informações do TClientDataSet clicando com o botão direito e escolhendo a opção Clear Data e em seguida, um componente TXMLTransformProvider deve ser adicionado no formulário. Este componente tem a funcionalidade de realizar o mapeamento em tempo real entre o XML padrão e o XML datapacket, utilizando para isso os arquivos gerados no tópico anterior. As seguintes propriedades devem ser configuradas:

TransformRead->TransformationFile: ToDp.xtr

TransformWrite->TransformationFile: ToXml.xtr

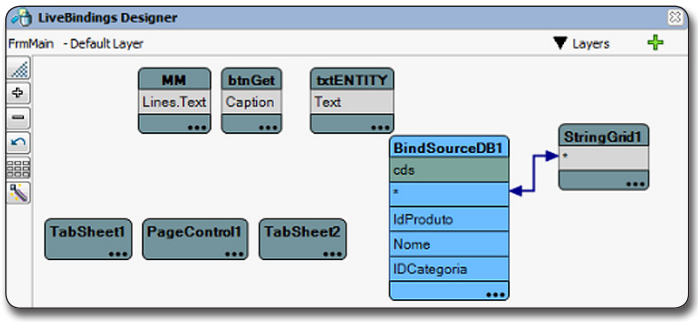

Figura 6. Visual Live Bindings entre o ClientDataSet e o StringGrid

A primeira está relacionada ao arquivo de transformação de XML padrão para XML datapacket, ou seja, dos dados vindos do servidor WCF para o Delphi; enquanto que a segunda faz o contrário; que é transformar os dados do Delphi para o servidor WCF. Além disso, o TClientDataSet deve apontar sua propriedade ProviderName para o XMLTransformProvider.

Depois de feitas essas configurações, as implementações da **Listagem 7** devem ser realizadas no evento OnClick do botão.

Listagem 7. Acesso aos produtos via Indy

```
01 procedure TfrmMain.btnGetProdutosClick(Sender: TObject)
02 var
03   XML: string;
04 begin
05   cdsProdutos.Close();
06   XML := IdHTTP.Get(URI + 'PRODUTOS');
07   prvProdutos.TransformRead.SourceXml := XML;
08   cdsProdutos.Open();
09 end;
```

Note que o procedimento é praticamente o mesmo realizando anteriormente, com as diferenças de que estamos informando o nome da entidade PRODUTOS como uma constante no método HTTP Get (linha 06). Além disso, o XMLTransformProvider recebe a fonte de dados que vem diretamente do servidor WCF através da propriedade SourceXml (linha 07).

O próximo passo é adicionar três TLabels, três TEdits e um TBindNavigator no formulário, os quais terão a função de mostrar os dados e possibilitar a navegação nos registros. A **Figura 7** mostra como ficarão os componentes ligados pelo recurso de Visual Live Bindings.

Vale lembrar que utilizando essa arquitetura, não são necessários controles do tipo Dataware do Delphi, tais como TDBEdits e TDataSource.

A **Figura 8** mostra o resultado final da tela com os componentes adicionados.

Configurando a aplicação mestre-detalhe

A última etapa de nosso exemplo é realizar a ligação mestre-detalhe entre as tabelas de categorias e produtos. Para isso é necessário seguir todas as etapas do mapeamento do arquivo XML vistas anteriormente, porém, agora utilizando o XML de CATEGORIAS.

Em seguida, um novo TClientDataSet e um TXMLTransformProvider devem ser adicionados no formulário, tendo suas propriedades ligadas igualmente ao XML de PRODUTOS. Por fim, os demais componentes visuais devem ser adicionados, bem como um TButton e uma TStringGrid. Mais uma vez o Visual Live Binding deve ser realizado entre o novo TClientDataSet e a TStringGrid recém adicionada, como mostrado anteriormente para o grid de produtos. A **Figura 9** mostra a disposição dos componentes na tela.

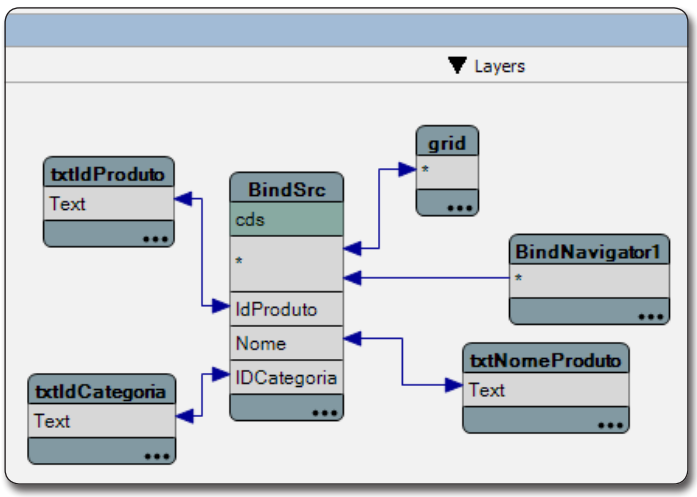

Figura 7. Visual Live Bindings entre os componentes

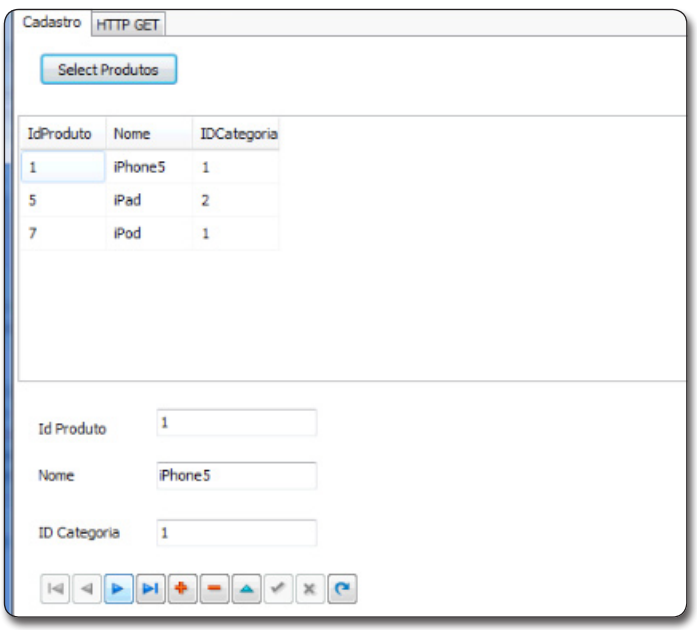

Figura 8. Navegação nos registros

A seguir, buscaremos do servidor WCF os dados das categorias, utilizando o código mostrado na **Listagem 8**. Note que o processo é idêntico ao anterior, havendo apenas algumas mudanças nos nomes dos componentes.

Para finalizar, o código sofrerá uma pequena alteração que busca os produtos, para que ao clique do botão sejam retornados no grid somente os produtos relativos à categoria selecionada no grid superior. A **Listagem 9** mostra o novo código após modificado.

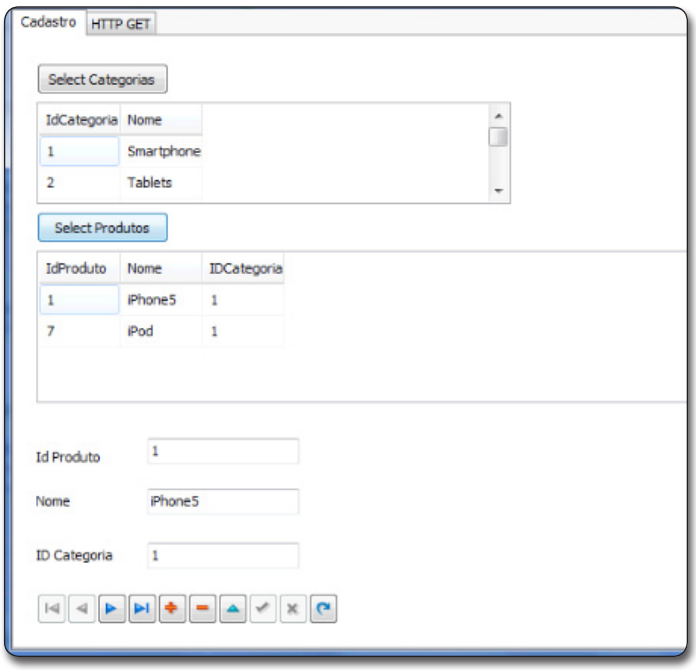

Figura 9. Mestre-detalhe

Listagem 8. Acesso ás categorias via Indy

```
01 procedure TFrmMain.btnGetCategoriasClick(Sender: TObject)
02 var
03   XML: string;
04 begin
05   cdsCategorias.Close();
06   XML := IdHTTP.Get(URI + 'CATEGORIAS');
07   prvCategorias.TransformRead.SourceXml := XML;
08   cdsCategorias.Open();
09 end;
```

Listagem 9. Filtrando os produtos pela categoria

```
01 procedure TFrmMain.btnGetProdutosClick(Sender: TObject);
02 var
03   XML: string;
04   filter: string;
05 begin
06   cdsProdutos.Close();
07   filter := '?$filter=IDCategoria%20eq%20' +
       cdsCategoriasIdCategoria.Value;
08   XML := IdHTTP.Get(URI + 'PRODUTOS' + filter);
09   prvProdutos.TransformRead.SourceXml := XML;
10   cdsProdutos.Open();
11 end;
```

O código da **Listagem 9** apresenta uma solução bastante interessante para filtrar os dados, sendo passado diretamente por meio da URL e sem a necessidade de criar um novo método no servidor para realizar um filtro nos produtos por categoria (condição where). Note que são usadas as palavras "filter" e "eq" (linha 07). A primeira é utilizada para indicar ao servidor WCF que existe

um filtro e a segunda equivale ao "igual", ou seja, o campo IdCategoria dever ser igual ao valor selecionado no TClientDataSet de categorias.

A **Figura 10** exibe a arquitetura final da solução.

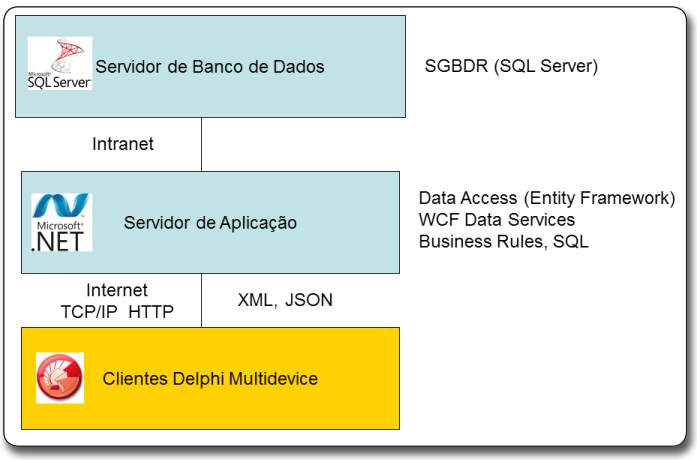

Figura 10. Arquitetura final da solução

Autor

Guinther Pauli
http://www.devmedia.com.br/guintherpauli
Arquiteto de Software certificado pela Microsoft, Consultor, Instrutor, Autor, Palestrante, Professor de Graduação e de Pós-Graduação, Certificado oficial Microsoft MCP, MCAD, MCSD.NET, MCTS, MCPD (ASP.NET, C# e Arquitetura) - Certificado oficial Delphi: 3, 5, 6, 7, 2005, 2006, Delphi para Web, Kylix.

Autor

Jones Granatyr
Doutorando em Informática bolsista da CAPES e Mestre em Informática bolsista do CNPq, ambos na área de Inteligência Artificial. Possui Especialização em Segurança em Redes de Computadores e Banco de Dados Distribuídos e Graduação em Sistemas de Informação. Trabalha em projetos de pesquisa relacionados a área Inteligência Artificial, tais como Sistemas Especialistas, Mineração de Dados, Mineração de Textos e Sistemas Multiagente.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DÊ UM SALTO EM CONHECIMENTO!

Acesse o maior portal para desenvolvedores da América Latina!

20 mil posts

430 mil cadastrados

10 milhões de page-views por mês

http://www.devmedia.com.br/

DEVMEDIA

Delphi XE2: Cloud Computing e Windows Azure

Interaja com Windows Azure Blob Storage através do Delphi XE2

Fique por Dentro

Este artigo explica o que é Cloud Computing e através de um exemplo mostra como acessar o Windows Azure Blob Storage através de aplicações desenvolvidas no Delphi XE2. Sabendo que o Windows Azure oferece um excelente ambiente para armazenamento de dados, serão explorados seus conceitos e vantagens. Em seguida, será abordada a API Cloud do Delphi, assim como seus componentes para interagir com Windows Azure Blob Storage. O exemplo criado permite o envio e recebimento de arquivos da nuvem, funcionando como uma cópia de segurança para informações.

A computação na nuvem (Cloud Computing) está cada vez mais presente no nosso dia-a-dia e vem moldando um novo modelo de TI para grandes, médias e pequenas empresas. Existem várias razões para essa enorme adesão, porém, antes de enumera-las é fundamental compreender que a Cloud Computing é um conjunto de serviços disponibilizados através da internet, onde você pode escolher quando e como utilizá-los.

Agora que entendemos o princípio fundamental da Cloud Computing podemos enumerar os principais fatores para seu sucesso, sendo eles: potencial "infinito" de armazenamento; possibilidade de executar aplicações em outros computadores através da internet de forma transparente; e por fim, seu custo benefício compatível. Esses benefícios permitem às empresas manter o foco em seus negócios e não se preocupar em gerenciar, por exemplo, Cluster milionários ou ambientes de desenvolvimento com centenas de licenças.

Para facilitar o entendimento, a Cloud Computing foi dividia em três camadas: Software como Serviço (SaaS), Plataforma como Serviço (PaaS) e Infraestrutura como Serviço (IaaS).

Camadas da Cloud Computing

A **Figura 1** apresenta as três camadas estruturais da Cloud Computing, Software como Serviço (SaaS), Plataforma como Serviço (PaaS) e Infraestrutura como Serviço (IaaS), e como estão distribuídos os serviços e aplicações mais utilizados.

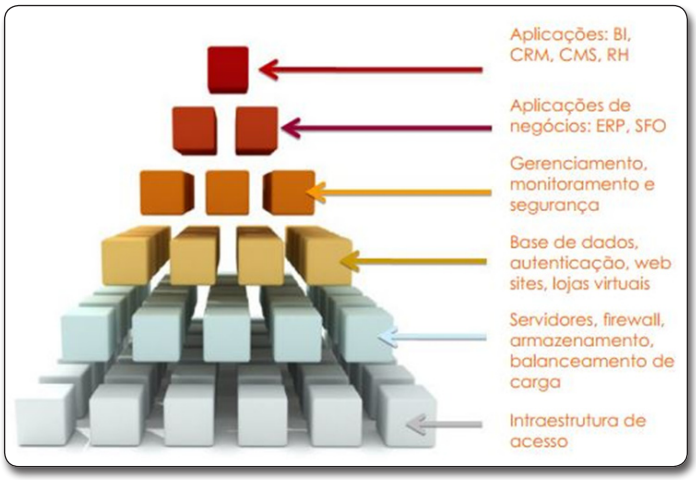

Figura 1. Camadas Cloud Computing

Software como Serviço (SaaS)

É um modelo de entrega de Software no qual o Software e seus dados associados são hospedados na internet (nuvem) e são acessados pelos usuários através de internet, normalmente usando um navegador web.

Utilizamos SaaS em praticamente todos os serviços de internet hoje em dia, como busca na Web (Google, Yahoo) ou seu Webmail (Gmail, Hotmail). De forma geral o conceito de Software como serviço é muito mais abrangente e mais aplicado no contexto de aplicações de negócios. A adoção da Cloud Computing e particularmente na modalidade de SaaS, é uma excelente forma de entregar rapidamente soluções tecnológicas para atender as necessidades das empresas.

Plataforma como Serviço (PaaS)

É a entrega de um ambiente de computação em camadas de soluções como serviço. Ofertas PaaS facilitam a implantação de aplicações de menor custo e complexidade na compra e gestão do hardware, software e recursos de provisionamento

de infraestrutura, que fornece todas as facilidades necessárias para suportar o ciclo de vida completo de construção e entrega de aplicações web e serviços totalmente disponíveis a partir da Internet.

O PaaS é um ambiente para a criação, hospedagem e controle de software. Ele pode ser oferecido como:

• Serviços de colaboração em equipe; Integração de banco de dados, persistência e gerenciamento de estado;
• Serviços de hospedagem (Web sites, Blogs, Lojas virtuais, etc.);
• Desenvolvimento, trazendo as facilidades para o Design da aplicação, controle de versão do aplicativo, testes e outros.

Há uma série de maneiras diferentes para caracterizar PaaS, mas algumas das características mais comuns encontradas em diversos provedores são:

• Ambiente para desenvolver, testar, implantar e manter aplicações de forma integrada e escalável, para cumprir todo o processo de desenvolvimento;
• Escalabilidade, incluindo balanceamento de carga e failover (processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha);
• Integração com serviços web e bases de dados através de padrões comuns;
• Segurança integrada.

O PasS é útil em qualquer situação onde vários desenvolvedores estarão trabalhando em um projeto, ou quando outras partes externas precisam interagir com o processo de desenvolvimento. E para atender essas necessidades a Microsoft oferece a plataforma Windows Azure.

Infraestrutura como Serviço (IaaS)

É uma maneira de entregar computação em nuvem, onde a infraestrutura de servidores, sistemas de rede, armazenamento e todo o ambiente necessário para o funcionamento são contratados como serviços. Ao invés de comprar servidores, Software, espaço em Data Center, os clientes usam estes recursos como um serviço totalmente terceirizado sob demanda.

Em IaaS, existem as subcategorias que são: nuvens públicas, privadas ou uma combinação das duas (híbrida).

Nuvem pública: Infraestrutura disponível para contratação, consiste em recursos compartilhados, padronizados e com auto-atendimento pela Internet.

Nuvem privada: Infraestrutura que utiliza as características da computação em nuvem, como a virtualização, mas na forma de uma rede privada.

Nuvem híbrida: É a combinação das nuvens públicas e privadas.

Assim como acontece com as duas camadas anteriores, o SaaS e o PaaS, o IaaS passam por um crescimento acelerado. As ofertas de soluções, os custos, a alta disponibilidade e o nível de profissionalização dos provedores são os principais fatores da adoção do IaaS, assim como o fato de seus recursos serem contratados como um serviço.

Plataforma Windows Azure

A plataforma Windows Azure oferece uma plataforma como serviço (PaaS) na nuvem fornecida pela Microsoft, podemos defini-la como um conjunto de tecnologias que nos permite executar serviços, armazenar objetos de dados (Storage), desenvolver e publicar aplicações de quaisquer tamanhos nos Data Centers da Microsoft distribuídos pelo globo, de forma escalar, mesmo você sendo uma grande empresa ou não.

A plataforma Windows Azure é dividida em quatro grupos:

• **Windows Azure**: Oferece recursos para computação, hospedagem de aplicações e serviços, gerenciamento e armazenamento de dados, através do Azure Storage;
• **SQL Azure**: Oferece recursos para relatórios, persistência de dados relacional e gerenciamento de dados na nuvem;
• **Windows Azure AppFabric**: Oferece dispositivos para os desenvolvedores conectarem aplicativos e serviços nas nuvens e local;
• **Windows Azure DataMarket**: Oferece mecanismos para gerenciamento de pagamentos, informações sobre serviços publicados e relatórios para o controle de aplicações oferecidas na nuvem.

Estes recursos podem ser consumidos por aplicações na nuvem ou local, conforme é apresentado na **Figura 2**.

Figura 2. Plataforma Windows Azure

Windows Azure

O Windows Azure oferece três grupos principais de funcionalidades para a execução de processos, aplicações e serviços num modelo de nuvem, são eles:

Computação: através de uma malha de máquinas virtuais disponíveis para contratação, o Windows Azure oferece um poder de processamento ilimitado para aplicações que exigem alto volume de computação.

Armazenamento: aplicações executadas na nuvem exigem tipos específicos preparados para uma alta escalabilidade e resposta. Ao mesmo tempo, aplicações na nuvem podem exigir um espaço ilimitado e crescente para o armazenamento de objetos diversos, como dados não estruturados, arquivos, figuras, imagens, vídeos, áudios, etc.

Gerenciamento: para aplicações e processos executados na plataforma Windows Azure, temos ferramentas de administração através das capacidades de gerenciamento da plataforma.

A seguir iremos explorar o Azure Storage, que é fornecido como parte das capacidades de armazenamento do Windows Azure.

Azure Storage

O Azure Storage é um componente importante da plataforma Windows Azure e é um ambiente completo para armazenamento de todos os tipos de informações: dados relacionais, imagens, vídeos, áudios, PDFs e etc.

A **Figura 3** apresenta uma visão geral do esquema de armazenamento proporcionado pelo Windows Azure.

Como podemos observar existem três tipos fornecidos pelo Azure Storage para armazenamento, este artigo irá abordar o tipo Blob, conforme será visto mais adiante.

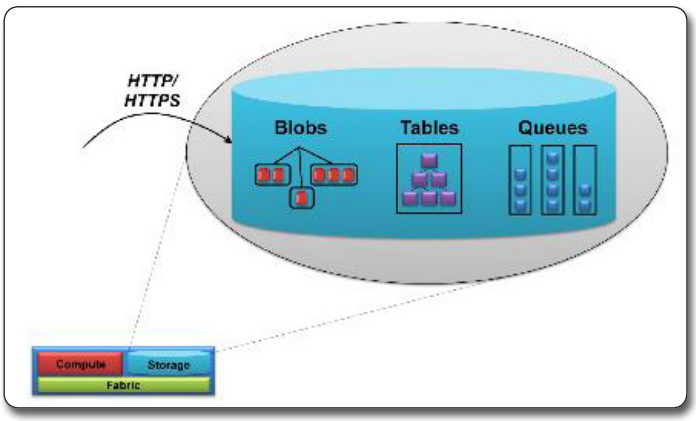

Figura 3. Modelo armazenamento Azure Storage

Windows Azure Blob Storage

No nosso dia-a-dia estamos familiarizados com o armazenamento de arquivos utilizando a estrutura de diretórios, com o Windows Azure Blob Storage não é diferente. Os arquivos são armazenados como binários ou textos dentro de containers e que ficam fisicamente em Blocks.

Quando a plataforma utiliza esta metodologia, o arquivo é dividido em n blocos de mesmo tamanho. Estes arquivos podem conter tags (metadados) que descrevem o arquivo ou adicionam informações sobre seu conteúdo. Cada container pode conter um ou mais arquivos de qualquer tipo especificado, com permissões específicas, podendo ser público ou privado. Os containers estão associados a sua conta de Storage no Windows Azure. A **Figura 4** ilustra a estrutura de blobs no Azure Storage.

Windows Azure no Delphi XE2

O Windows Azure disponibiliza aos desenvolvedores APIs para acessar os serviços de armazenamento (Blob, Table e Queue - (**ver BOX 1 e BOX 2**)) via REST (**ver BOX 3**). Para acessar esses recursos o Delphi disponibilizou na versão XE uma API que interage com os serviços de armazenamento do Windows Azure. A partir da versão XE2, a API do Delphi foi redesenhada e passou a chamar AzureAPI e não mais DSAzure, assim os componentes visuais (TAzureBlobManagagement, TAzureQueueManagement, TAzureTableManagement) que também foram considerados obsoletos.

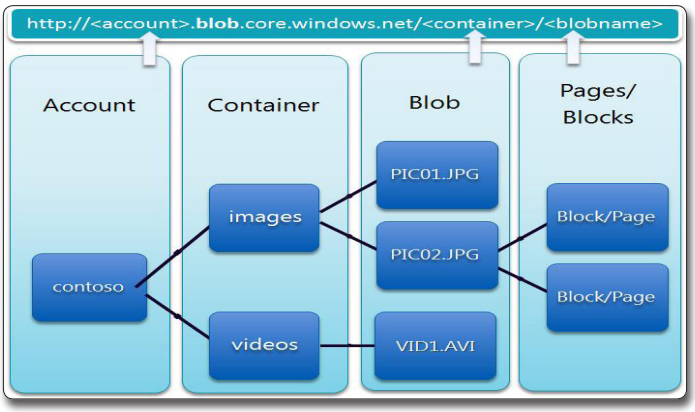

Figura 4. Modelo armazenamento dos blobs

BOX 1. Windows Azure Table

Windows Azure Table proporciona um ambiente estruturado na forma de tabelas que agrupam objetos sem estrutura definida. A ideia é que estas estruturas distintas encontrem-se armazenadas em tabelas onde, cada linha corresponde a um novo objeto e cada coluna corresponda as propriedades deste objeto.

BOX 2. Windows Azure Queue

Windows Azure Queue (Fila) é mais um dos recursos de armazenamento disponibilizados pela plataforma Azure. O principal objetivo deste serviço é oferecer um sistema de comunicação assíncrono altamente escalável e disponível.

BOX 3. REST

REST (Transferência de Estado Representacional) pode ser definido como um estilo de projetar aplicativos Web fracamente acoplados, cujos recursos são nomeados e acessíveis diretamente através de uma URI (Identificador Uniforme de Recursos) exclusiva para cada recurso.

Azure API

A Azure API oferece suporte nativo aos serviços Azure Blobs, Azure Tables e Azure Queue e é através da unit Data.Cloud.AzureAPI que temos acesso as classes que implementam esta API. A seguir temos uma breve descrição das principais classes ilustradas na **Figura 5**.

- TAzureService: É a classe base para todas as classes de serviço do Windows Azure. Esta fornece funcionalidade comum para Queues, Tables e Blobs e implementa a classe base de serviço em nuvem TCloudService.
- TAzureBlobService: Implementa a classe base TAzureService, serve para gerenciar uma conta do Windows Azure Blob Service no Delphi.
- TAzureQueueService: Implementa a classe base TAzureService, serve para gerenciar uma conta do Windows Azure Table Service no Delphi.
- TAzureTableService: Implementa a classe base TAzureService, serve para gerenciar uma conta do Windows Azure Queue Service no Delphi.

• TAzureConnectionInfo: Descende da classe TCloudConnectionInfo, fornece informações como as definições de armazenamento de desenvolvimento, cria uma instância de conexão com o Windows Azure e recebe as URLs de serviço. Esta classe possui componente não visual e está localizada na sessão Cloud da Tool Palette.

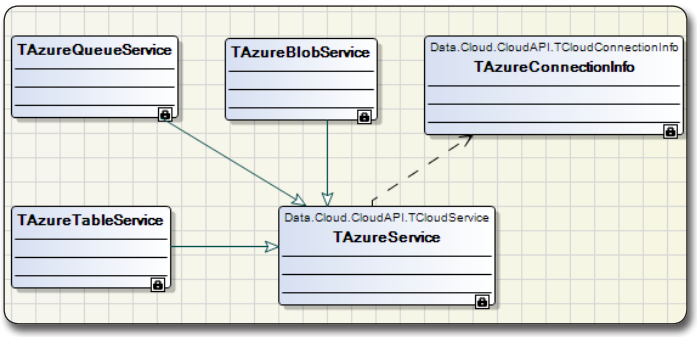

Figura 5. Diagrama de classe simplificado das principais classes da AzureAPI

Windows Azure Blob Storage no Delphi

Como visto anteriormente, um conjunto de classes forma a API para acesso ao Azure Storage. Podemos destacar a classe TAzureBlobService herdando de TAzureService. Esta classe recebe uma instância de TAzureConnectionInfo configurada com os dados da conexão no construtor e provê métodos para enumerar os Blobs, criar novos, listar os Containers e gravar ou obter informações deles. Outra classe que merece destaque é a TCloudResponseInfo que é retornada pela maioria dos métodos de TAzureBlobService, este objeto retornado contém o status referente a solicitação feita.

Exemplo prático - Acessando o Windows Azure Blob Storage através do Delphi XE2

Após conhecermos melhor os conceitos envolvidos na Cloud Computing, suas tecnologias, vantagens e a API disponibilizada pelo Delphi para acesso a camada de armazenamento no Windows Azure, iremos iniciar um exemplo que demonstrará como criar Containers, enviar arquivos e baixar arquivos no Widows Azure Blob Storage.

Criando uma conta no Windows Azure

Para esta aplicação devemos possuir uma conta ativa no Windows Azure, a mesma pode ser adquirida facilmente através de seu site (vide seção **Links**) com promoção de 30 dias de avaliação gratuita (na data de puplicação deste artigo). O Windows Azure disponibiliza diversos serviços, mas para este exemplo iremos precisar apenas de contratar o serviço de armazenamento, pois, o componente TAzureConnectionInfo precisa dos parâmetros AccountName (nome da conta de armazenamento) e AccountKey (chave de acesso primária ou secundária), conforme **Figura 6**.

Estrutura da aplicação

Nossa aplicação foi desenvolvida utilizando o Delphi XE2 com Hotfix 6 e será constituída de duas classes: TCloudAzure e TFrmCloudAzure, a classe TCloudAzure será responsável por toda a comunicação com o Windows Azure, como conectar-se ao serviço de armazenamento, enviar arquivos para a nuvem, criar containers, criar metadados (tanto para os containers quanto para os arquivos), obter informações e baixar os arquivos para o disco conforme apresentado na **Figura 7**. Como não é intenção deste artigo esgotar todos os recursos da API Azure do Delphi, não serão abordados os métodos para deletar e alterar Containers e arquivos.

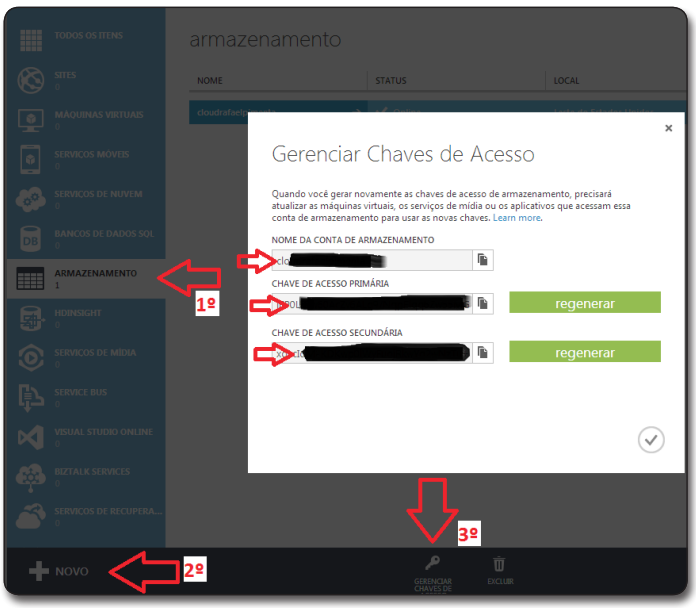

Figura 6. Painel de configuração dos serviços do Windows Azure

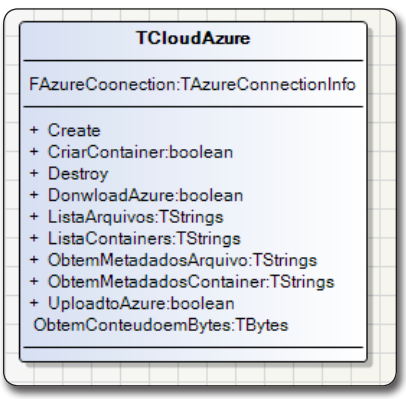

Figura 7. Representação da classe TCloudAzure

Nota

As bibliotecas OpenSSL (libeay32.dll e ssleay32.dll) são necessárias para autenticar solicitações no Windows Azure através do Delphi.

Iniciando conexão com o Windows Azure Blob Storage

O método Create da classe TCloudAzure (**Listagem 1**) recebe como parâmetros o nome da conta e a chave (linha 01). Após isso, é instanciado um objeto do tipo TAzureCoonectionInfo (linha 03).

Como já explicado, este objeto cria uma instância de conexão com o Windows Azure e sempre que formos nos comunicar com o Windows Azure iremos utilizá-lo. Entre as linhas 04 a 06 temos a configuração das demais propriedades (utilizando os parâmetros).

Listagem 1. Método Create da classe TCloudAzure

```
01 constructor TCloudAzure.Create(NomeConta, ChaveConta: string);
02 begin
03   FAzureCoonection := TAzureConnectionInfo.Create(nil);
04   FAzureCoonection.UseDevelopmentStorage := False;
05   FAzureCoonection.AccountName := NomeConta;
06   FAzureCoonection.AccountKey := ChaveConta;
07 end;
```

Listando os containers e seus metadados existes no Windows Azure Blob Storage

O método ListaContainers é responsável por retornar uma lista do tipo TStringList com os nomes de todos containers existentes na conta do Windows Azure. Com esse tipo de retorno (TString-List), tornamos flexível a escolha do componente que irá exibir estas informações, podendo ser um TStringGrid, TListView entre outros. A **Listagem 2** exibe o código responsável pelo método.

Na linha 8 é instanciado um objeto do tipo TAzureBlobService, este objeto é quem nos fornece os métodos para gerenciar os dados armazenados em nossa conta, como o método ListContainers. Que por sua vez, retorna uma lista de TAzureContainer (linha 13), pode-se notar que, assim como a grande maioria dos métodos da classe TAzureBlobService, este recebe como parâmetro um objeto do tipo TCloudResponseInfo. Nas linhas 17 e 18 é realizado um loop no objeto ContainerList (lista de TAzureContainer) e para cada laço populamos o Result do método com o nome do container, porém, antes verificamos através do objeto ResponseInfo se a lista de Container foi retornada com sucesso (linha 15), o significado de cada StatusCode pode ser visto através do capítulo Status Code Definitions do documento Hypertext Transfer Protocol HTTP/1.1 (ver seção **Links**).

O método ObtemMetadadosContainer mostrado na **Listagem 3** recebe o nome de um container como parâmetro e é responsável por retornar uma lista do tipo TStringList com os metadados (informações adicionais) deste container.

Para obter os metadados seguimos o mesmo princípio que foi utilizado na lista de container: instaciamos um objeto do tipo TAzureBlobService (linha 07), porém, agora iremos fazer uso do método GetContainerMetadata (linha 11). Conforme pode ser notado, este método recebe como parâmetro o nome do container (NomeContainer), um TStrings (Result) que na verdade é uma parâmetro por referência do tipo OUT, ou seja, ele é quem irá nos retornar a lista de metadados, e um TCloudResponseInfo (ResponseInfo). A seguir na linha 12 é testado o retorno, exibindo uma exceção caso seja diferente de 200.

Listagem 2. Método ListaContainers da classe TCloudAzure

```
01 function TCloudAzure.ListaContainers: TStrings;
02 var
03   BlobService: TAzureBlobService;
04   ContainerObj: TAzureContainer;
05   ContainerList: TList<TAzureContainer>;
06   ResponseInfo: TCloudResponseInfo;
07 begin
08   BlobService := TAzureBlobService.Create(FAzureCoonection);
09   ResponseInfo := TCloudResponseInfo.Create;
10   Result := TStringList.Create;
11   try
12     try
13       ContainerList := BlobService.ListContainers(nil, ResponseInfo);
14       try
15         if ResponseInfo.StatusCode <> 200 then // 200 = OK
16           raise Exception.Create(ResponseInfo.StatusMessage);
17         for ContainerObj in ContainerList do
18           Result.Add(ContainerObj.Name);
19       finally
20         ContainerList.Free;
21       end;
22     except
23       on E: Exception do
24       begin
25         raise Exception.Create(E.Message);
26       end;
27     end;
28   finally
29     FreeAndNil(ResponseInfo);
30     FreeAndNil(BlobService);
31   end;
32 end;
```

Listagem 3. Método ObtemMetadadosContainer da classe TCloudAzure

```
01 function TCloudAzure.ObtemMetadadosContainer(NomeContainer: string):
02   TStrings;
03 var
04   BlobService: TAzureBlobService;
05   ResponseInfo: TCloudResponseInfo;
06 begin
07   BlobService := TAzureBlobService.Create(FAzureCoonection);
08   ResponseInfo := TCloudResponseInfo.Create;
09   Result := TStringList.Create;
10   try
11     BlobService.GetContainerMetadata(NomeContainer,Result,ResponseInfo);
12     if ResponseInfo.StatusCode <> 200 then // 200 = OK
13       raise Exception.Create(ResponseInfo.StatusMessage);
14   finally
15     FreeAndNil(ResponseInfo);
16     FreeAndNil(BlobService);
17   end;
18 end;
```

Listando os arquivos e seus metadados existentes no Windows Azure Blob Storage

A **Listagem 4** descreve a implementação do método ListaArquivos. Este recebe o nome de um Container como parâmetro de entrada, retornando uma lista do tipo TStringList com os nomes dos arquivos deste Container.

Este método segue a mesma lógica apresentada no método GetContainerMetadata (linha 11 da **Listagem 3**). A novidade neste método fica por conta da chamada à função ListBlobs

(linha 13) da classe TAzureBlobService. Ela retorna uma lista do tipo TAzureBlob, que nada mais é que nossa lista com os nomes dos arquivos. Entre as linhas 17 e 18, a lista de Blobs é percorrida e os respectivos nomes são incluídos ao StringList (Result).

Listagem 4. Método ListaArquivos da classe TCloudAzure

```
01 function TCloudAzure.ListaArquivos(NomeContainer: string): TStrings;
02 var
03  BlobService: TAzureBlobService;
04  BlobObj: TAzureBlob;
05  BlobList: TList<TAzureBlob>;
06  ResponseInfo: TCloudResponseInfo;
07 begin
08  BlobService  := TAzureBlobService.Create(FAzureCoonection);
09  ResponseInfo := TCloudResponseInfo.Create;
10  Result := TStringList.Create;
11  try
12   try
13    BlobList := BlobService.ListBlobs(NomeContainer,nil,ResponseInfo);
14    try
15     if ResponseInfo.StatusCode <> 200 then // 200 = OK
16      raise Exception.Create(ResponseInfo.StatusMessage);
17     for BlobObj in BlobList do
18      Result.Add(BlobObj.Name);
19    finally
20     BlobList.Free;
21    end;
22   except
23    on E: Exception do
24    begin
25     raise Exception.Create(E.Message);
26    end;
27   end;
28  finally
29   FreeAndNil(ResponseInfo);
30   FreeAndNil(BlobService);
31  end;
32 end;
```

Na **Listagem 5** é exibido o método ObtemMetadadosBlob que recebe como parâmetros o nome de um container e de um arquivo contido nele. Este é responsável por retornar uma lista do tipo TStringList com os metadados (informações adicionais) deste arquivo.

A única diferença deste método para o ObtemMetadadosContainer encontram-se nas linhas 11 e 12, pois aqui é chamado o método GetBlobMetadata. Este recebe como parâmetro o nome do container, nome do arquivo, um TStrings (Result) que irá nos retornar a lista de metadados e por último um TCloudResponseInfo (ResponseInfo), que como já visto, é utilizado para validar se tudo está OK.

Criando um novo container e seus metadados no Windows Azure Blob Storage

O método CriarContainer (**Listagem 6**) como o próprio nome sugere, é responsável por criar um novo Container na conta do Windows Azure, seu retorno é do tipo booleano e nos informa se o mesmo foi ou não criado com sucesso. Na linha 12 é chamado o CreateContainer, que por sua vez, recebe como parâmetro o nome do novo container, os metadados, o item (bpaContainer) do tipo enumerado TBlobPublicAccess que define a visibilidade do container e de seus arquivos dele como público, e por fim, um objeto do tipo TCloudResponseInfo. Na linha 14 é verificado através do objeto ResponseInfo se o container foi criado com sucesso.

Listagem 5. Método ObtemMetadadosBlob da classe TCloudAzure

```
01 function TCloudAzure.
    ObtemMetadadosArquivo(NomeContainer, NomeArquivo:
02  string): TStrings;
03 var
04  BlobService: TAzureBlobService;
05  ResponseInfo: TCloudResponseInfo;
06 begin
07  BlobService := TAzureBlobService.Create(FAzureCoonection);
08  ResponseInfo := TCloudResponseInfo.Create;
09  Result := TStringList.Create;
10  try
11   BlobService.GetBlobMetadata(NomeContainer,NomeArquivo,Result,
12    '',ResponseInfo);
13   if ResponseInfo.StatusCode <> 200 then // 200 = OK
14    raise Exception.Create(ResponseInfo.StatusMessage);
15  finally
16   FreeAndNil(ResponseInfo);
17   FreeAndNil(BlobService);
18  end;
19 end;
```

Listagem 6. Método CriarContainer da classe TCloudAzure

```
01 function TCloudAzure.CriarContainer(NomeContainer: string;
02  Metadados: TStrings = nil): boolean;
03 var
04  BlobService  : TAzureBlobService;
05  ResponseInfo : TCloudResponseInfo;
06 begin
07  BlobService  := TAzureBlobService.Create(FAzureCoonection);
08  ResponseInfo := TCloudResponseInfo.Create;
09  Result := False;
10  try
11   try
12    BlobService.CreateContainer(NomeContainer, Metadados,
13     bpaContainer,ResponseInfo);
14    if not(ResponseInfo.StatusCode in [200, 201]) then //OK, CREATED
15     raise Exception.Create(ResponseInfo.StatusMessage)
16    else
17     Result := True;
18   except
19    on E: Exception do
20    begin
21     raise Exception.Create(E.Message);
22    end;
23   end;
24  finally
25   FreeAndNil(BlobService);
26   FreeAndNil(ResponseInfo);
27  end;
28 end;
```

Enviando arquivo e seus metadados para o Windows Azure Blob Storage

Na **Listagem 7** é apresentado o método UploadAzure, responsável por enviar um arquivo selecionado para dentro de um container na conta do Windows Azure. Seu retorno é do tipo booleano e nos informa se o arquivo foi enviado com sucesso ou não.

Listagem 7. Método UploadAzure da classe TCloudAzure

```
01 function TCloudAzure.UploadAzure(FileName, NomeContainer,
02   NomeArquivo: String; Metadados: TStrings = nil): boolean;
03 var
04   BlobService: TAzureBlobService;
05   ResponseInfo: TCloudResponseInfo;
06   Content: TBytes;
07 begin
08   BlobService := TAzureBlobService.Create(FAzureCoonection);
09   ResponseInfo := TCloudResponseInfo.Create;
10   Result := False;
11   if (FileName = EmptyStr) then
12     Exit;
13   try
14     try
15       Content := ObtemConteudoEmBytes(FileName);
16       BlobService.PutBlockBlob(NomeContainer,NomeArquivo,Content, '',
17         nil, Metadados, ResponseInfo);
18       if not(ResponseInfo.StatusCode in [200, 201]) then //OK, CREATED
19         raise Exception.Create(ResponseInfo.StatusMessage)
20       else
21         Result := True;
22     except
23       on E: Exception do
24       begin
25         raise Exception.Create(E.Message);
26       end;
27     end;
28   finally
29     FreeAndNil(BlobService);
30     FreeAndNil(ResponseInfo);
31   end;
32 end;
```

Na linha 15 é chamado o método ObtemConteudoEmBytes que recebe como parâmetro o endereço do arquivo no Windows (FileName) e retorna o conteúdo do arquivo em bytes para a variável Content (TBytes). Em seguida, é chamado o método PutBlockBlob (linhas 16 e 17) que envia o conteúdo do arquivo juntamente com seus metadados para denro de um cotainer. Note que mais uma vez é utilizado o valor de Response para validar a resposta.

Baixando arquivos do Windows Azure Blob Storage para o disco

O método DownloadAzure (**Listagem 8**) é responsável por baixar um arquivo da conta do Windows Azure para o disco rígido, seu retorno é do tipo booleano, indicando se o arquivo foi ou não baixado com sucesso.

Na linha 17 chamamos o método GetBlob que recebe como parâmetros o nome do Container que se encontra o arquivo, o nome do arquivo a ser baixado, um uma variável do tipo TStream (que irá armazenar o conteúdo do arquivo) e um objeto do tipo TCloudResponseInfo. Na linha 19 instanciamos a variável FS (TFileStream) passando o endereço onde será salvo o arquivo. Por fim, posicionamos a Stream no seu início (linha 20) e copiamos seu conteúdo para a variável FS, responsável por criar o arquivo em disco (linha 21).

Listagem 8. Método DownloadAzure da classe TCloudAzure

```
01 function TCloudAzure.DownloadAzure(NomeContainer, NomeArquivo,
02   FilePath: String): boolean;
03 var
04   BlobService: TAzureBlobService;
05   ResponseInfo: TCloudResponseInfo;
06   Stream: TStream;
07   FS: TFileStream;
08 begin
09   BlobService := TAzureBlobService.Create(FAzureCoonection);
10   ResponseInfo := TCloudResponseInfo.Create;
11   Stream := TMemoryStream.Create;  //conterá o arquivo baixado
12   Result := False;
13   if (FilePath = EmptyStr) then
14     Exit;
15   try
16     try
17       BlobService.GetBlob(NomeContainer, NomeArquivo, Stream, '',
18         ResponseInfo);
19       FS := TFileStream.Create(FilePath + NomeArquivo, fmCreate);
20       Stream.Position := 0;
21       FS.CopyFrom(Stream, Stream.Size);
22       if not(ResponseInfo.StatusCode in [200, 201]) then
23         raise Exception.Create(ResponseInfo.StatusMessage)
24       else
25         Result := True;
26       FS.Free;
27     except
28       on E: Exception do
29       begin
30         raise Exception.Create(E.Message);
31       end;
32     end;
33   finally
34     FreeAndNil(BlobService);
35     FreeAndNil(ResponseInfo);
36     FreeAndNil(Stream);
37   end;
38 end;
```

Consumindo a classe TCloudAzure através do formulário

A classe TCloudAzure será consumida pelo formulário TFrmCloudAzure, o objetivo deste formulário é prover os recursos disponíveis da classe TCloudAzure de forma gráfica, conforme apresentado na **Figura 8**.

O formulário possui a simples tarefa de instanciar um objeto TCloudAzure e efetuar chamadas aos seus métodos nos eventos dos componentes do formulário. O formulário exibido na **Figura 8** é constituído dos seguintes componentes:

- frmCloudAzure: TfrmCloudAzure;
- gbxContainers (TGroupBox)
  - listViewContainers (TListView);
  - vleMetaContainer (TValueListEditor);
  - btnCriarContainer (TButton);
- gbxArquivos (TGroupBox);
  - ListViewArquivos (TListView);
  - vleMetaArquivo (TValueListEditor);
  - btnUploadArquivo (TButton);
  - btnDownloadArquivo (TButton);

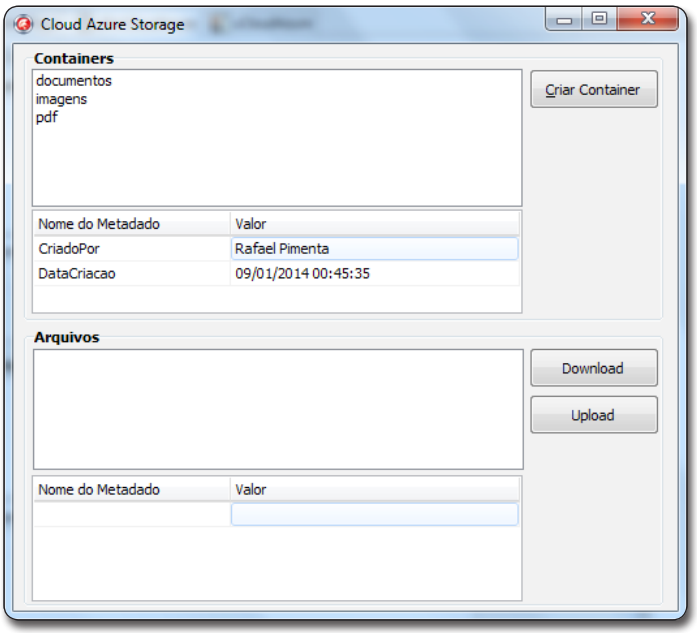

Figura 8. Formulário principal da aplicação TFrmCloudAzure

Através deste artigo foi possível explorar os principais conceitos e vantagens envolvidos na computação em nuvem, exploramos de forma técnica - através da API Cloud do Delphi - uma pequena fatia de todo o seu potencial, que é o serviço de armazenamento em nuvem Windows Azure Blob Storage. Este recurso se mostrou bastante interessante para situações onde aplicações Desktop, Web ou Mobile pretendem acessar e armazenar diversos tipos de arquivos na nuvem, sem se preocupar com capacidade e disponibilidade, pois através do mesmo existe a possibilidade de criar repositórios de código-fonte, bibliotecas de música, vídeos, imagens, entre outros.

Mesmo tendo pontos a melhorar, como a necessidade de conexão com a Internet, uma infinidade de possibilidades surge com a Cloud Computing e os cenários atuais e futuros apontam para isto. Esperamos que este artigo estimule o leitor a aprofundar seus estudos a cerca desse novo modelo de TI e seus recursos, que vão muito além do armazenamento de dados.

Autor

Rafael Santos Pimenta
Rafa.spimenta@gmail.com

Programador e analista de sistemas utilizando tecnologia Delphi a mais de 5 anos. Graduado em Ciência da Computação pela faculdade Pitágoras, possui certificação Delphi Developer e atualmente trabalha como analista de sistemas no departamento de tecnologia da PUC Minas.

Links:

Introdução sobre o Windows Azure
http://msdn.microsoft.com/pt-br/library/hh150078.aspx

Documentação Delphi - Desenvolvimento de aplicativos em nuvem
http://docwiki.embarcadero.com/RADStudio/XE2/en/Developing_Cloud_Applications

Portal do Windows Azure
https://www.windowsazure.com/

OpenSSL
http://www.openssl.org/

Status Code Definitions - Hypertext Transfer Protocol -- HTTP/1.1
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Somos tão apaixonados por tecnologia que o nome da empresa diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,
assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

Ele está trabalhando em seu saque e não na sua nova aplicação.

Mais de 95%* dos times de desenvolvimento e testes de aplicativos relatam tempos de espera e atrasos para acessar os sistemas que necessitam para realizar suas atividades. A Virtualização de Serviços elimina estas dependências gerando ambientes simulados similares à realidade, permitindo o desenvolvimento em paralelo. Isto significa que suas aplicações serão lançadas mais rapidamente, com maior qualidade e menor custo. Game over!

Conheça mais em ca.com/br/GoDevOps

*De acordo com o estudo 2012 North America/Europe Service Virtualization Study, Coleman-Parkes.

ca technologies